Technische Universität München
TUM School of Engineering and Design

Kopplung von Graphersetzung und parametrischer Modellierung zur Unterstützung des modellbasierten Entwerfens und der Erstellung mehrskaliger Modelle

Simon Felix Franziskus Vilgertshofer

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. Pierluigi D'Acunto

Prüfer der Dissertation:

1. Prof. Dr.-Ing. André Borrmann
2. Prof. Dr. Timo Hartmann
3. Prof. Dr.-Ing. Frank Petzold

Die Dissertation wurde am 22.09.2022 bei der Technischen Universität München eingereicht

und durch die TUM School of Engineering and Design am 28.11.2022 angenommen.

Zusammenfassung

Die vorliegende Dissertation befasst sich mit der Unterstützung und Teilautomatisierung komplexer und arbeitsintensiver Modellierungsprozesse von dreidimensionalen parametrischen Bauwerksmodellen. Dazu wird ein Lösungsansatz vorgestellt, der es ermöglicht, die Abfolge von konsekutiven oder auch konkurrierenden Modellierungsoperationen formalisiert zu definieren und bei Bedarf abzurufen und automatisiert auszuführen. So können Modellierungsabläufe formalisiert und in verschiedenen Modellierungsszenarien angewendet werden. Weiterhin ist es so möglich, die Qualität der digitalen Modelle zu verbessern, da durch die Anwendung der formalisierten Modellierungsschritte Fehler vermeidbar werden, die bei einer rein manuellen Modellierung auftreten können.
Zur formalen Definition und Speicherung der Modellierungsoperationen werden Graphersetzungsregeln eingesetzt. Diese Graphersetzungsregeln sind so konzipiert, dass nicht nur eine Änderung und Erweiterung der geometrischen Elemente erfolgt, sondern auch die parametrischen Zusammenhänge des Modells entsprechend der Regeldefinition verändert und erweitert werden. Auch nach einer graphbasierten Modellerstellung können die Modelle dementsprechend über die Veränderung der Parameterwerte an veränderte Randbedingungen angepasst werden. Eine wesentliche Voraussetzung für die Definition und Anwendung von Graphersetzungsregeln zur formalen Definition der Modellierungsoperationen besteht in einer graphbasierten Repräsentation der parametrischen Modelle, da nur ein in Form eines Graphen gespeichertes parametrisches Modell auch durch eine Graphersetzungsregel verändert werden kann. Daher ist die Erarbeitung eines Konzepts zur graphbasierten Repräsentation parametrischer Modelle ein wesentlicher Bestandteil dieser Arbeit.

Abstract

This dissertation addresses the support and partial automation of complex and labor-intensive modeling processes of three-dimensional parametric building models. For this purpose, an approach is presented, which makes it possible to define a sequence of consecutive or even competing modeling operations in a formalized way and to retrieve and automatically execute them on demand. Thus, modeling sequences can be formalized and applied in different modeling scenarios. Furthermore, it is thus possible to improve the quality of the digital models, since the application of the formalized modeling steps makes it possible to avoid errors that can occur with solely manual modeling.

Graph rewriting rules are used to formally define and store the modeling operations. These graph rewriting rules are designed to not only change and extend the geometric elements, but also to change and extend the parametric constraints of the model according to the rule definition. Consequently, even after a graph-based model creation, models can be adapted to changed boundary conditions by changing the parameter values. An essential prerequisite for the definition and application of graph substitution rules for the formal definition of modeling operations is a graph-based representation of the parametric models, since only a parametric model stored in the form of a graph can also be modified by a graph rewriting rule. Therefore, the development of a concept for the graph-based representation of parametric models is an essential part of this work.

Vorwort

Die vorliegende Arbeit ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Computergestützte Modellierung und Simulation an der Technischen Universität München im Zeitraum vom Oktober 2014 bis September 2022 entstanden. An dieser Stelle möchte ich mich bei all jenen herzlich bedanken, die mich während dieser Zeit unterstützt und damit zum Gelingen dieser Arbeit beigetragen haben.

In erster Linie gilt mein Dank Prof. Dr.-Ing. André Borrmann für die Anregung zu dieser Arbeit und die stets ausgezeichnete Betreuung. Neben den regelmäßigen Diskussionen zu wissenschaftlichen Fragestellungen möchte ich dabei insbesondere die kontinuierliche Motivation, sein großes Interesse am Thema der Arbeit und die mir entgegengebrachte Geduld unterstreichen.

Des Weiteren gilt mein Dank Prof. Dr. Timo Hartmann, Prof. Dr.-Ing. Frank Petzold und Prof. Dr. Pierluigi D'Acunto für die Bereitschaft den Abschluss dieser Arbeit als Gutachter bzw. Vorsitzender der Prüfungskommission zu begleiten.
Außerdem möchte ich mich bei Prof. Dr. rer. nat. Ernst Rank bedanken, der bereits während des Grundstudiums mein Interesse für die Bauinformatik geweckt hat.

Für das Gelingen dieser Arbeit war auch das inspirierende, motivierende und diskussionsfreudige Arbeitsumfeld an der Technischen Universität München von wesentlicher Bedeutung. Daher möchte ich mich bei allen Kolleginnen und Kollegen am Lehrstuhl für Computergestützte Modellierung und Simulation für das hervorragende und stets kollegiale Miteinander bedanken.

Auch bedanken möchte ich mich bei meiner Familie und bei meinen Freundinnen und Freunden, die mich immer wieder aufs Neue motiviert haben. Dieser Dank gilt in erster Linie meinen Eltern, die mich auf meinem Weg zur und durch die Promotion stets unterstützt haben. Vor allem erwähnen möchte ich dabei meine Mutter Jutta, die einen ganz wesentlichen Beitrag zur sprachlichen und orthografischen Korrektheit dieser Arbeit geleistet hat.

Schließlich möchte ich mich ganz besonders bei Martina bedanken, die mir während der doch mitunter recht arbeitsreichen Promotionszeit den nötigen Rückhalt gegeben und mir insbesondere in der letzten Phase die nötige Zeit eingeräumt hat, diese Arbeit erfolgreich abzuschließen.

Simon Vilgertshofer
München, Dezember 2022

Inhaltsverzeichnis

1 Einführung **1**
1.1 Ausgangspunkt und Motivation 2
1.2 These, Ziel und Fragestellungen 4
1.3 Prämissen 5
1.4 Aufbau der Arbeit 7
1.5 Publikationen im Rahmen dieser Arbeit 8

2 Hintergrund **9**
2.1 Einsatz von Modellen zum Entwurf von Gebäuden und Bauwerken 10
2.1.1 Der Modellbegriff 11
2.1.2 Produktmodelle im Bauwesen 15
2.1.3 Geometrische Modellierung von Produktmodellen 20
2.1.4 Einordnung 24
2.2 Detaillierung von Modellen 25
2.2.1 Level of Detail 26
2.2.2 Detaillierungsgrade im Bauwesen 30
2.2.3 Mehrskalige Modelle 36
2.3 Ansätze zur computerorientierten Unterstützung von Entwurfs- und Konstruktionsprozessen 40
2.3.1 Automatisierung von Modellierungs- und Entwurfsprozessen 41
2.3.2 Graphbasierte Ansätze zur Automatisierung von Modellierungs- und Entwurfsprozessen 45
2.4 Zusammenfassung und Einordnung im Kontext dieser Arbeit 48

3 Geometrische und parametrische Modellierung **49**
3.1 Geometrische Modellierung von Volumenmodellen 50
3.1.1 Explizite Verfahren 51

3.1.2 Implizite Verfahren . . . 53
3.1.3 Einordnung . . . 56
3.2 Parametrische Modellierung . . . 57
3.2.1 Grundlagen . . . 59
3.2.2 Struktur und Historie eines parametrischen Modells . . . 63
3.2.3 Parametrische Zwangsbedingungen . . . 67
3.2.4 Constraint Solver . . . 68

4 Graphen und Graphersetzungssysteme 74
4.1 Grundbegriffe der Graphentheorie . . . 75
4.1.1 Definitionen . . . 77
4.1.2 Visualisierung von Graphen . . . 84
4.2 Graphersetzung . . . 85
4.2.1 Überblick . . . 87
4.2.2 Graphersetzungsregeln . . . 90
4.2.3 Graph-Metamodell . . . 94
4.2.4 Graphmustersuche . . . 95
4.2.5 Ansätze zur Graphersetzung . . . 96
4.2.6 Software Frameworks zur Graphersetzung . . . 98

5 Elemente eines parametrischen Modellierungsprozesses als Gegenstand der graphbasierten Repräsentation 102
5.1 Umfang der abzubildenden parametrischen Modelle . . . 103
5.2 Grundlegende Eigenschaften der Objekte eines parametrischen Modells . . . 105
5.3 Objekte auf Baugruppenebene . . . 105
5.4 Objekte auf Bauteilebene . . . 108
5.5 Objekte auf 2D-Skizzenebene . . . 111
5.5.1 Geometrische Elemente . . . 112
5.5.2 Parametrische Zwangsbedingungen . . . 118
5.6 Integration von Semantik . . . 125
5.7 Zusammenfassung . . . 126

6 Ein Graphersetzungssystem zur Automatisierung parametrischer Modellierungsoperationen 127
6.1 Vorgehensweise und Entwicklungsprozess . . . 129
6.1.1 Konzept . . . 130
6.1.2 Graph und Modell . . . 132
6.2 Konzeption des Graphen . . . 136
6.2.1 Eigenschaften des Graphen . . . 137
6.2.2 Hierarchischer Aufbau des Graphen . . . 139
6.2.3 Verwendung von Knoten und Kanten . . . 143

6.2.4 Andockstellen . . . 145
6.2.5 Temporäre Koordinaten . . . 148
6.2.6 Nicht explizit im Graphen enthaltene Geometrie . . . 150
6.3 Formale Definition des Graphen . . . 154
6.4 Graph-Metamodell . . . 156
6.4.1 Überblick über den Aufbau des Graph-Metamodells . . . 158
6.4.2 Implementierung des Graph-Metamodells . . . 162
6.4.3 Beispielhafte graphbasierte Repräsentationen . . . 174
6.5 Graphersetzungsregeln . . . 182
6.5.1 Definition und Aufbau von Graphersetzungsregeln zur Repräsentation von Modellierungsoperationen . . . 184
6.5.2 Beispiele . . . 186
6.6 Technische Umsetzung . . . 201
6.7 Zusammenfassung . . . 204

7 Fallstudien 205
7.1 Modellierung eines Tunnels in verschiedenen Detaillierungsgraden . . . 206
7.2 Tunnelquerschlag und Objekt in LoD 5 . . . 218
7.2.1 Querschlag . . . 218
7.2.2 Objekt im LoD 5 . . . 224
7.3 Modellierung von Profilträgern . . . 227
7.4 Zusammenfassung . . . 231

8 Zusammenfassung, Diskussion und Ausblick 232
8.1 Diskussion der Einsatzmöglichkeiten und Grenzen . . . 233
8.2 Ausblick . . . 236

A Definition des Graph-Metamodells in GrGen.NET 239

B Weitere Graphersetzungsregeln 244

Literaturverzeichnis 259

Kapitel 1

Einführung

In den 90er Jahren fand mit der Einführung des Computer Assisted Design (CAD) ein wesentlicher Schritt zur Digitalisierung der Planungsprozesse im Bauwesen statt. Während technische Zeichnungen bis dato vornehmlich mit großem manuellen Aufwand auf Papier handgezeichnet wurden, konnten nun Änderungen in der Planung durch die Nutzung von CAD-Softwareprogrammen schneller und einfacher in die Zeichnungen eingefügt werden. Auch wenn die Umstellung vom papier- zum computerbasierten Arbeiten anfangs sicherlich mit einem zusätzlichen Aufwand verbunden war, lässt sich in Retrospektive klar feststellen, dass dieser Digitalisierungsprozess insgesamt vorteilhaft war.

Einen ähnlichen Paradigmenwechsel stellt das modellbasierte Arbeiten dar. Die Abkehr vom rein zeichnungsbasierten Arbeiten etablierte sich im Jahrzehnt vor Veröffentlichung dieser Arbeit sowohl national als auch global immer weiter. Inzwischen werden in einer Vielzahl von Bauprojekten digitale 3D-Modelle eingesetzt, die als Grundlage für verschiedene nachfolgende Anwendungsfälle dienen können und damit zu einer Verbesserung der Planungsqualität und einer Reduzierung der Planungs- und Baukosten beitragen. Nichtsdestoweniger müssen auch diese Bauwerksmodelle manuell modelliert werden, was einen erhöhten Anspruch an die Fähigkeiten und das Wissen der Nutzer stellt.

Die vorliegende Arbeit befasst sich mit der Unterstützung und Teilautomatisierung komplexer und arbeitsintensiver Modellierungsprozesse von dreidimensionalen parametrischen Bauwerksmodellen. Dazu wird ein Lösungsansatz vorgestellt, der es ermöglicht, die Abfolge von konsekutiven oder auch konkurrierenden Modellierungsschritten formalisiert zu definieren und bei Bedarf abzurufen und automatisiert auszuführen. So können Modellierungsabläufe formalisiert und in verschiedenen Modellierungsszenarien angewendet werden.

Weiterhin ist es so möglich, auch die Qualität der digitalen Modelle zu verbessern, da durch die Anwendung der formalisierten Modellierungsschritte Fehler vermieden werden, die bei einer rein manuellen Modellierung auftreten können.

Zur formalen Definition und Speicherung der Modellierungsoperationen werden Graphersetzungsregeln eingesetzt. Diese Graphersetzungsregeln sind so konzipiert, dass nicht nur eine Änderung und Erweiterung der geometrischen Elemente erfolgt, sondern auch die parametrischen Zusammenhänge des Modells entsprechend der Regeldefinition verändert und erweitert werden. Auch nach einer graphbasierten Modellerstellung können die Modelle dementsprechend über die Veränderung der Parameterwerte an veränderte Randbedingungen angepasst werden. Eine wesentliche Voraussetzung für die Definition und Anwendung von Graphersetzungsregeln zur formalen Definition der Modellierungsoperationen besteht in einer graphbasierten Repräsentation der parametrischen Modelle, da nur ein in Form eines Graphen gespeichertes parametrisches Modell auch durch eine Graphersetzungsregel verändert werden kann. Daher ist auch die Erarbeitung eines Konzepts zur graphbasierten Repräsentation parametrischer Modelle Bestandteil dieser Arbeit.

1.1 Ausgangspunkt und Motivation

Die erfolgreiche Planung und Ausführung von Bauprojekten ist ein komplexer Prozess, der sich durch viele verschiedene Beteiligte aus unterschiedlichen Fachdisziplinen auszeichnet. Während selbst kleine und einfache Projekte komplexe Fragen aufwerfen können, ist dies bei großen Projekten sehr häufig der Fall.

Dies bedingt sich durch die Vielzahl verschiedener Randbedingungen und Einschränkungen genauso wie durch die Kooperation einer großen Anzahl von Projektbeteiligten aus verschiedenen Fachgebieten. In Laufe des letzten Jahrzehnts haben sich durch die immer weitere Verbreitung der Methode des Building Information Modeling (BIM) teils wesentliche Veränderungen in altbewährten Arbeitsabläufen mit bedeutendem Einfluss auf die einzelnen Fachdisziplinen und ihre Zusammenarbeit und Kommunikation untereinander ergeben.

Eine zentrale Rolle nimmt hier die Verwendung detaillierter dreidimensionaler digitaler Bauwerksmodelle ein, die neben der rein geometrischen Repräsentation des Bauwerks und seiner Einzelteile zusätzlich semantische Informationen beinhalten. Diese Modelle sind die Grundlage für viele Konzepte und Anwendungsmöglichkeiten, die im Kontext von BIM erforscht und entwickelt, sowie in der Baupraxis eingeführt und umgesetzt werden. Szenarien wie eine kooperative Bauplanung (Borrmann, 2007), die automatisierte Überwachung des Baufortschritts (Braun, 2020), eine modellbasierte Überprüfung der Einhaltung von Vorschriften und Richtlinien (Preidel, 2020), die BIM-gestützte Kostenanalyse oder die Generierung und Optimierung von Bauzeitplänen auf Basis von Modellen (Dori, 2016; Sigalov

et al., 2017) wären ohne entsprechende Modelle, die verschiedene Arten von Informationen enthalten, nicht möglich. Die Erstellung hochwertiger Modelle ist somit eine wesentliche Voraussetzung für die oben genannten Anwendungsszenarien sowie für eine Vielzahl weiterer Anwendungsfälle von BIM.

Mit der Weiterentwicklung von BIM und den damit einhergehenden zusätzlichen Anwendungsmöglichkeiten steigen jedoch auch die Anforderungen an die notwendigen digitalen Bauwerksmodelle. Dies betrifft gleichermaßen die mit der Modellierung beauftragten Projektbeteiligten und entsprechend auch die von ihnen eingesetzten Softwarewerkzeuge zur Erstellung der Modelle: Neben der Möglichkeit, die Modelle mit semantischen Informationen anzureichern, muss auch die Geometrie in Bezug auf Detaillierung, Flexibilität und Konformität korrekt erstellt werden können. Dies bedeutet in vielen Fällen einen erhöhten Aufwand bei der Erstellung der digitalen Modelle.

Grundsätzlich ist es jedoch wünschenswert, den manuellen Arbeitsaufwand der Modellierung gering zu halten, um Zeit und Kosten bei der Erstellung komplexer geometrischer Modelle zu reduzieren und den beteiligten Fachplanern die Freiräume zu verschaffen, sich mit den wesentlichen Ingenieuraufgaben in Entwurf und Konstruktion auseinanderzusetzen. Die verschiedenen Techniken im Bereich der parametrisch-assoziativen Modellierung, zu dem auch die parametrische Modellierung gehört, tragen bereits wesentlich zur Erleichterung des Modellierungsprozesses bei. Modelle können kontinuierlich flexibel angepasst werden, Änderungen lassen sich schnell umsetzen und die Modelle werden trotzdem konsistent gehalten.

Allerdings erfordert die Erstellung solcher parametrisch-assoziativer Modelle einerseits entsprechende Fach- und Softwarekenntnisse bei den Anwendern und ist weiterhin mit einem hohen manuellen Aufwand verbunden. Konkret bedeutet dies, dass die Entwicklung von Methoden, die die Erzeugung dieser Modelle unterstützen oder automatisieren, einen wesentlichen Beitrag zur Effizienzsteigerung bei der Erstellung digitaler Bauwerksmodelle leisten kann. Einerseits können repetitive Tätigkeiten so zumindest teilweise automatisiert werden. Darüber hinaus wird die Grundlage dafür geschaffen, das einem spezifischen Modellierungsergebnis inhärente Expertenwissen abzubilden und später wieder abzurufen.

Der in dieser Arbeit beschriebene Ansatz soll einen Beitrag zur Entwicklung solcher Methoden leisten. Der Ansatz bedient sich der Methode der Graphersetzung und untersucht damit eine Vorgehensweise zur Erstellung parametrischer Modelle, die bisher noch nicht Gegenstand der wissenschaftlichen Forschung war. Das übergeordnete Ziel der Arbeit besteht dabei in der Entwicklung einer Methode, die es ermöglicht Experten- und Konstruktionswissen formal abzubilden und damit eine automatisierte Ausführung von Modellierungsvorgängen zu ermöglichen.

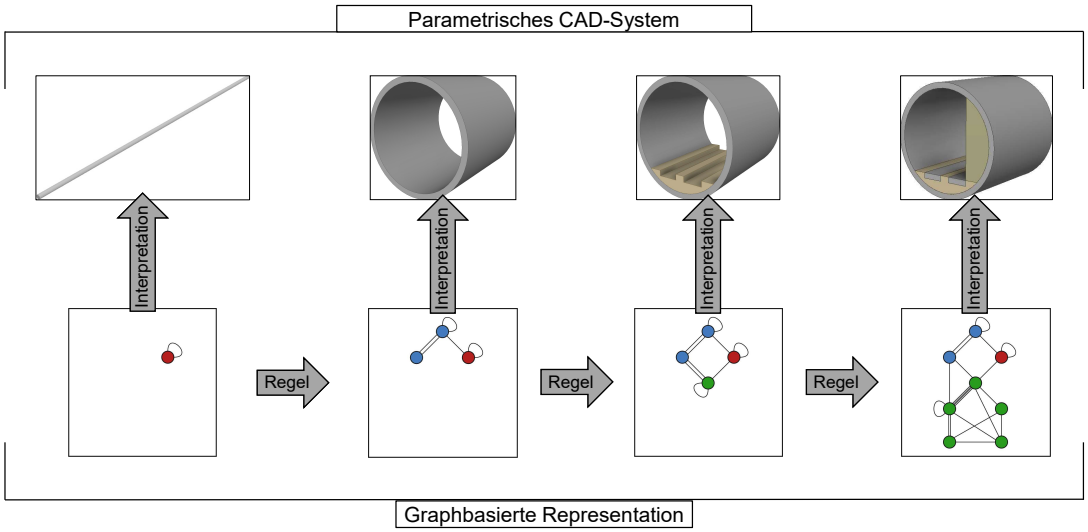


Abbildung 1.1: Beispielhafter Modellierungsablauf eines Tunnelquerschnitts. Die einzelnen Modellierungsschritte werden durch Graphersetzungsregeln abgebildet; der so erstellte Graph kann nach jedem Schritt interpretiert werden, um ein bearbeitbares parametrisches Modell zu erzeugen.

1.2 These, Ziel und Fragestellungen

Die These, die in dieser Arbeit vertreten, ausgearbeitet und diskutiert wird, besagt, dass ein parametrischer Modellierungsprozess mithilfe eines Graphersetzungssystems formal beschrieben werden kann. Einzelne Modellierungsschritte werden dabei durch Graphersetzungsregeln formalisiert und können zu einem späteren Zeitpunkt wiederholt ausgeführt und bei Bedarf auch in einem anderen Kontext angewendet werden. Dies bildet die Grundlage für eine Reduzierung des manuellen Arbeitsaufwands bei der Erstellung von digitalen Bauwerksmodellen. Da repetitive Tätigkeiten bei der Modellerstellung durch die Anwendung von bereits definierten Regeln automatisiert ausgeführt werden können, ist es außerdem für die Planer möglich, sich stärker auf wesentliche Entwurfsentscheidungen zu fokussieren.

Dies ist insbesondere bei der schrittweisen Detaillierung eines parametrischen Modells während eines voranschreitenden Entwurfsprozesses hilfreich. Dies bedingt sich dadurch, dass auch bei scheinbar vollkommen unterschiedlichen Planungsaufgaben einzelne Bauteile und damit auch die zur Modellierung und Detaillierung notwendigen Schritte projektübergreifend wiederholt werden können. Zusätzlich kann das zur Ausführung dieser Detaillierungsschritte nötige Expertenwissen in einem Graphersetzungssystem produktneutral abgebildet werden, sodass das entsprechende Vorgehen bei der Modellierung unabhängig von einem bestimmten Modellierungswerkzeug erhalten bleibt. Damit kann sichergestellt werden, dass auch bei einem Wechsel des Modellierungswerkzeugs die grundlegende Logik eines parametrischen Modells nicht verloren geht. Abbildung 1.1 zeigt den grundlegenden Ansatz anhand eines beispielhaften Modellierungsablaufs.

Ziel dieser Arbeit ist die Untersuchung von Graphrepräsentationen und Graphersetzungsverfahren auf Eignung zur Abbildung von Modellerstellungsvorgängen und aufbauend darauf die Entwicklung verschiedener Graphersetzungssysteme, die die hierfür notwendigen Funktionalitäten enthalten und mit denen anhand verschiedener Beispiele gezeigt werden kann, dass der entwickelte Ansatz praktisch umsetzbar ist.
Aus diesem übergeordneten Ziel ergeben sich die folgenden wesentlichen Fragestellungen:

- Wie können parametrische Modelle durch einen Graph repräsentiert werden, sodass sie durch ein Graphersetzungssystem formal verändert werden können?
- Wie können einzelne oder mehrere Schritte eines parametrischen Modellierungsprozesses durch Graphersetzungsregeln, aus denen sich dieses Graphersetzungssystem zusammensetzt, beschrieben werden?
- Welchen Anforderungen müssen der das Modell repräsentierende Graph und das Graphersetzungssystem genügen, um sicherzustellen, dass durch die Interpretation des Graphen ein nachträglich bearbeit- und veränderbares parametrisches Modell in verschiedenen CAD-Anwendungen erzeugt werden kann, um eine praktische Anwendung zu ermöglichen?
- Wo liegen die Grenzen dieser Vorgehensweise und des Graphersetzungssystems hinsichtlich der Umsetzbarkeit, des Aufwands und des Nutzens in der praktischen Anwendung?

1.3 Prämissen

Dem in dieser Arbeit entwickelten Ansatz liegen zwei wesentliche Annahmen zugrunde, die im Folgenden einführend dargestellt werden.

Dies betrifft einerseits die Entscheidung, die Technik der parametrischen Modellierung[1] zu nutzen. die im Folgenden kurz als parametrische Modellierung bezeichnet wird. Damit wird nicht das tatsächliche Ergebnis einer Modellierung – also die Geometrie eines Modells – repräsentiert, sondern die einzelnen Schritte des Modellierungsprozesses und damit die logischen Zusammenhänge, aus denen sich dieses Ergebnis ergibt. Grundsätzlich gibt es eine Vielzahl von Methoden und Vorgehensweisen, die zur computergestützten Speicherung und Erstellung digitaler Modelle verwendet werden können. Ein entsprechender Überblick über dieses Themenfeld wird in Kapitel 3 gegeben, wobei insbesondere zwischen impliziten und expliziten Geometrierepräsentationen unterschieden wird.

[1] Der Ausdruck *parametrische Modellierung* wird hier als Oberbegriff für die verschiedenen Methoden der parametrisch-assoziativen Modellierung, der historienbasierten prozedural-parametrischer Modellierung und der variationalen Modellierung benutzt (Obergrießer, 2016; Shah *et al.*, 1995). Eine differenziertere Betrachtung der verschiedenen Konzepte der parametrischen Modellierung ist in Kapitel 3 enthalten.

Der wesentliche Unterschied zwischen diesen Methoden liegt darin, dass bei einem expliziten Modell nur die finale Geometrie als Ergebnis eines Modellierungsprozesses gespeichert wird (Romberg, 2005; Wesley *et al.*, 1984). Ein implizites Modell enthält hingegen die einzelnen Konstruktionsschritte, die zu einem Ergebnis führen – aber nicht zwangsläufig die konkrete finale Geometrie (Obergrießer, 2016), da dieser erst aus der impliziten Modellbeschreibung berechnet werden muss (Neuberg, 2004). Bei einer expliziten Repräsentation eines Modells gehen die Informationen zum Ablauf der Modellierung hingegen in der Regel weitgehend verloren. Damit einher geht auch ein Verlust des Wissens über diesen Ablauf.

Ein parametrisches Modell definiert sich nun genau dadurch, dass das Ergebnis der Modellierung in Abhängigkeit der Parameterwerte einzelner Modellierungsoperationen nachträglich modifiziert werden kann. Daher muss ein solches Modell auch alle relevanten Informationen zu diesen Modellierungsoperationen erhalten. Durch diese Modifizierbarkeit bleiben parametrische Modelle während eines Planungsprozesses flexibel, sodass sich Änderungen in vielen Fällen mit vergleichsweise geringem Aufwand umsetzen lassen. Die Vorteile dieses Paradigmas zeigen sich auch darin, dass sich viele Softwarewerkzeuge zur Erstellung digitaler Bauwerksmodelle (sog. BIM-Autorenwerkzeuge) den Methoden der parametrischen Modellierung auf die ein oder andere Weise bedienen. Inzwischen ist der Einsatz der parametrischen Modellierung aus gängigen BIM-Autorenwerkzeugen kaum mehr wegzudenken, seit dessen Relevanz bereits vor über fünfzehn Jahren von Lee *et al.* (2006b) als wesentlich zur Einbettung von Expertenwissen in BIM-Modelle hervorgehoben wurde.

Neben der nachträglichen Modifizierbarkeit liegt ein weiterer wesentlicher Vorteil eines parametrischen Modells darin, dass der Vorgang der Modellierung nachvollzieh- und damit auch reproduzierbar bleibt. Soll nun – wie in dieser Arbeit – untersucht werden, wie sich die Erstellung von Modellen und damit auch der Vorgang des Entwerfens unterstützen und teils auch automatisieren lässt, liegt es nahe, parametrische Modelle zu verwenden. Dies bedingt sich zusätzlich durch die parametrischen Modellen inhärente Möglichkeit Änderungen an Teilen eines Modells automatisch auf das gesamte Modell zu propagieren. Anders ausgedrückt können parametrische Modelle – wenn sie denn korrekt und konsistent erstellt werden – allein durch ihre grundlegende Beschaffenheit dazu beitragen, einen von Änderungen geprägten Entwurfsprozess und damit auch den Modellierungsprozess zu vereinfachen und die Qualität des resultierenden Modells zu erhöhen. Mit der Entwicklung einer Methode, die den parametrischen Modellierungsprozess weiter unterstützt, macht man sich diese Vorteile bereits von Beginn an zu nutze.

Beispielhaft sind hier die Arbeiten von Borrmann *et al.* zu nennen, in denen die Modellierung von Schildvortriebstunneln in unterschiedlichen Detaillierungsgraden untersucht wurde (Borrmann *et al.*, 2014; Borrmann *et al.*, 2013a; Borrmann *et al.*, 2015; Borrmann *et al.*, 2013b). Um Inkonsistenzen zwischen den verschiedenen Detaillierungsgraden zu vermeiden, wurden parametrische Modellierungstechniken angewendet, die bei Änderungen die auto-

matische Erhaltung der Konsistenz des Modells über die verschiedenen Detaillierungsgrade hinweg ermöglichen. Allerdings haben die Ergebnisse von Borrmann *et al.* auch gezeigt, dass die manuelle Erstellung konsistenzerhaltender parametrischer Produktmodelle eine sehr komplexe, zeitaufwändige und fehleranfällige Aufgabe ist, die dringend einer Unterstützung durch computerbasierte Verfahren bedarf.

Die zweite Annahme besteht darin, dass Graphen optimal genutzt werden können, um die parametrischen Modelle zu repräsentieren. Dies bedingt sich darin, dass Graphen sehr gut zur Modellierung komplexer Systeme genutzt werden können (Heckel *et al.*, 2020) und dabei insbesondere die Zusammenhänge von Objekten in diesen Systemen formal repräsentieren. Da ein parametrisches Modell letztlich ein System von Zusammenhängen verschiedener Objekte darstellt, liegt die Verwendung von Graphen nahe – insbesondere auch, da Graphen schon erfolgreich zur Beschreibung von *Constraint-Problemen* im Kontext der parametrischen Modellierung eingesetzt werden (siehe Kapitel 3).

Der Kern dieser Annahme begründet sich allerdings nicht nur darin, dass Graphen geeignet sind, um Objekte und ihre Zusammenhänge zueinander vollständig zu beschreiben, da dies auch mit anderen Formen der Repräsentation möglich wäre. Vielmehr lassen sich über Graphersetzungsregeln die Veränderungen komplexer topologischer Zusammenhänge zwischen verschiedenen Objekten formal abbilden – wenn diese durch einen Graphen repräsentiert werden. Damit können sowohl der Graph zur Repräsentation eines parametrischen Modells als auch die Graphersetzung zur Repräsentation der Veränderungen des Modells als vorteilhaft angesehen werden. Eine weiterführende Auseinandersetzung mit dieser Entscheidung ist zu Beginn von Kapitel 6 aufgeführt, wobei die Vorteile und Möglichkeiten von Graphen in Kapitel 4 dargestellt werden.

1.4 Aufbau der Arbeit

Im folgenden Kapitel 2 wird zunächst der Hintergrund des Ansatzes in Bezug auf Anwendungsmöglichkeiten und den Stand der Forschung behandelt, um die Motivation für die Arbeit weiter auszuführen und eine Abgrenzung zu verwandten wissenschaftlichen Arbeiten vorzunehmen.

In den Kapiteln 3 und 4 werden die für das Verständnis dieser Arbeit notwendigen Grundlagen behandelt. Dazu wird zuerst auf die computergestützte geometrische und parametrische Modellierung eingegangen. Anschließend werden die grundlegenden Definitionen der Graphentheorie und der Aufbau von Graphersetzungssystemen besprochen. Beide Kapitel dienen dabei auch der einheitlichen Definition von Begriffen, die im weiteren Verlauf der Arbeit verwendet werden.

Im 5. Kapitel beginnt die Darlegung des Ansatzes, der im Rahmen dieser Arbeit entwickelt wurde. Hierfür wird zuerst der Umfang der Modellierungsoperationen, die durch das Graphersetzungssystem abgebildet werden sollen, beschrieben. Dabei werden die Anforderungen an das Graphersetzungssystem, die sich durch die zu repräsentierenden geometrischen Elemente, parametrischen Zwangsbedingungen und prozeduralen Operationen ergeben, definiert. Auf dieser Basis wird im folgenden Kapitel 6 die Entwicklung und der konkrete Aufbau des Graphersetzungssystems beschrieben und definiert. Zusätzlich zur abstrakten formalen Beschreibung werden hierbei auch einfache konkrete Beispiele zur besseren Verständlichkeit verwendet. Komplexere Beispiele und konkrete Fallstudien werden im sich anschließenden Kapitel 7 beschrieben.

Die Ergebnisse und Erkenntnisse der Arbeit werden im letzten Kapitel der Arbeit in Bezug auf ihre Grenzen und eine potenzielle Umsetzung in der Praxis diskutiert. Dieses Kapitel und die Arbeit enden mit abschließenden Betrachtungen, einer Zusammenfassung der Ergebnisse und einem Ausblick auf mögliche Weiterentwicklungen.

1.5 Publikationen im Rahmen dieser Arbeit

Teil der Ergebnisse dieser Arbeit wurden bereits in Konferenzbeiträgen und Zeitschriftenartikeln veröffentlicht. Innerhalb der vorliegenden Arbeit werden Teile dieser Artikel wiedergegeben.

Vilgertshofer, S. & Borrmann, A. (2015). Automatic detailing of parametric sketches by graph transformation. *Proc. of the 32nd International Symposium on Automation and Robotics in Construction and Mining*, Oulu, Finnland.

Vilgertshofer, S. & Borrmann, A. (2016a). A graph transformation based method for the semi-automatic generation of parametric models of shield tunnels. *23rd International Workshop of the European Group for Intelligent Computing in Engineering, EG-ICE 2016*, Kraków, Poland.

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.

Vilgertshofer, S. & Borrmann, A. (2018). Supporting feature-based parametric modeling by graph rewriting. *Proc. of the 35nd International Symposium on Automation and Robotics in Construction and Mining*, Berlin, Germany.

Kapitel 2

Hintergrund

Der im Rahmen dieser Arbeit entwickelten Methode liegen als Motivation und fachliche Basis eine Vielzahl von bereits bekannten bzw. ausführlich erforschten Themengebieten und entsprechenden wissenschaftlichen Veröffentlichungen zugrunde. In diesem Kapitel werden die verschiedenen Themengebiete und Vorarbeiten beschrieben, aus denen sich letztlich die Fragestellungen, die in dieser Arbeit untersucht werden, ergeben haben. Dies schließt auch die Einordnung dieser Arbeit in den Kontext bestehender Ansätze mit ein.

Zuerst wird dabei die Einführung und Entwicklung von (parametrischen) Produktmodellen im Bauwesen ganz generell ausgehend vom allgemeinen Modellbegriff beschrieben. Weiterhin wird vorgestellt, inwieweit mehrskalige Modelle im Zuge des Entwurfs von Bauwerken notwendig sind und wie diese entsprechend dem aktuellen Stand der Technik hinsichtlich ihrer Detaillierungsgrade definiert sind. Darauf aufbauend wird herausgearbeitet, weshalb bei der Erstellung solcher Modelle die Nutzung von parametrischen Modellierungswerkzeugen sinnvoll ist.

Anschließend werden Vorarbeiten betrachtet, aus denen sich die Motivation für die vorgeschlagene Nutzung von Graphersetzungssystemen zur Unterstützung eines parametrischen Modellierungs- und Entwurfsprozesses ableitet. Dazu werden verschiedene Methoden vorgestellt, die sich allgemein mit der Entwurfsautomatisierung beschäftigen und dabei teilweise auch Graphen und/oder Graphersetzungssysteme nutzen. Dabei wird auch die Thematik des Knowledge-based Engineering (KBE) aufgegriffen, um zu zeigen, dass die hier entwickelte Methode durch die graphbasierte und damit formale Repräsentation von Modellierungswissen zu diesem Forschungsfeld beitragen kann.

Das Kapitel schließt mit einer Einordnung der aufgeführten Themen hinsichtlich der Motivation und Zielsetzung dieser Arbeit ab.

2.1 Einsatz von Modellen zum Entwurf von Gebäuden und Bauwerken

Modelle werden in den verschiedensten Bereichen der Wissenschaft, der Technik und auch des täglichen Lebens genutzt, um die Realität vereinfacht abzubilden. Modelle können zur vereinfachten Abbildung von konkreten Objekten oder Informationen, genutzt werden. Außerdem können Theorien, mit denen Zusammenhänge in der wirklichen oder gedachten Welt verständlich ausgedrückt werden sollen, über Modelle beschrieben werden. Dies schließt konzeptionelle, mathematische und physische Modelle genauso wie architektonische Miniaturmodelle oder grafische Modelle mit ein.

Der Zweck dieser Modelle liegt beispielsweise darin, abstraktes oder konkretes Wissen, das durch das Modell repräsentiert wird zu transportieren – es also entweder auszutauschen oder lehrend zu vermitteln. Gerade bei der Vermittlung von wissenschaftlichen Zusammenhängen ist nach Harrison *et al.* (2000) ein Lehren oder Lernen ohne Modelle kaum möglich. Dies bedingt sich auch dadurch, dass Wissenschaft letztlich selbst als der Konstruktionsprozess von prädiktiven Modellen definiert werden kann, um Konzepte der realen Welt zu repräsentieren (Gilbert, 1991) und Modelle damit in der Wissenschaft von enormer Wichtigkeit sind (Leatherdale, 1974). In diesem Kontext sind Modelle aber als abstrakte gedankliche Konstrukte zu verstehen, denen nicht notwendigerweise ein reales greifbares Original zugrunde liegt, sondern ein Gedanke, eine Theorie oder ein erkannter Zusammenhang.

Demgegenüber stehen maßstäbliche Miniaturmodelle, wie sie beispielsweise in der Architektur eingesetzt werden (Petzold, 2004), die ein greifbares real existierendes Objekt repräsentieren. Dies schließt auch Objekte mit ein, die *noch* nicht real existieren, deren Existenz aber prinzipiell möglich ist. Modelle, die Objekte beschreiben, die perspektivisch existieren könnten, werden auch als präskriptive Modelle bezeichnet (Stachowiak, 1973). Während sich die Literatur zu Modellen in der Wissenschaft nur unwesentlich mit solchen maßstäblichen Modellen beschäftigt, stellen solche Modelle nichtsdestoweniger ein wichtiges Werkzeug für Naturwissenschaftler und Ingenieure dar, das bereits im 19. Jahrhundert in den Werken von Maxwell und Kelvin aufkommt (Leatherdale, 1974) und von Maxwell auch als „physikalische Analogie“ bezeichnet wird (Turner, 1955). Mit dem fortschreitenden Einsatz computergestützter Werkzeuge im Laufe des späten 20. Jahrhunderts spielen analoge Miniaturmodelle eine immer kleiner werdende Rolle in den Ingenieurwissenschaften, da ihnen digitale Modelle in vielerlei Hinsicht überlegen sind.

2.1.1 Der Modellbegriff

Trotz der im vorangegangenen Abschnitt beschriebenen ursprünglich differenzierten Verwendung des Begriffs *Modell*, können generelle Aussagen über den Modellbegriff getroffen werden, ohne dabei konsequent zwischen den verschiedenen Verwendungen zu unterscheiden. Im folgenden Abschnitt wird daher zuerst der Modellbegriff allgemein eingeführt und anschließend hinsichtlich seiner Verwendung im Kontext digitaler Modelle betrachtet.

Der Modellbegriff nach Stachowiak

Der Modellbegriff ist nach Stachowiak (1973) immer zumindest durch die drei folgenden Hauptmerkmale gekennzeichnet:

- Das **Abbildungsmerkmal** besagt, dass jedes Modell ein Modell von etwas, also eine Abbildung oder Repräsentation eines Originals ist. Dieses Original kann dabei auch selbst ein Modell sein. Auf Basis dieser Aussage ist es also möglich, ein Original stufenweise immer weiter zu abstrahieren, wodurch Modelle entstehen, die das Original in verschiedenen Abstraktionsgraden oder *Detaillierungsgraden* (siehe Abschnitt 2.2) repräsentieren. Eine Abstraktion kann aber auch aus verschiedenen Perspektiven erfolgen, über die ein Modell einen bestimmten Aspekt des Originals hervorhebt oder verdeutlicht. Das Original kann sowohl ein statisches Objekt als auch ein sich veränderndes System sein.

- Entsprechend dem **Verkürzungsmerkmal** beinhaltet ein Modell nicht alle Attribute oder Eigenschaften des Originals sondern nur diejenigen, die für den Verwendungszweck des Modells relevant sind. Daraus lässt sich ableiten, dass bei Verringerung des Detaillierungsgrads eines Modells durch Abstraktion weniger Attribute vorhanden sein können.

- Durch das **pragmatische Merkmal** wird festgelegt, dass Modelle ihren Originalen nicht automatisch eindeutig zugeordnet sind. Dies besagt, dass ein Modell seine repräsentierende Funktion immer für einen bestimmten Modellbenutzer zu einem bestimmten Zeitpunkt und zu einem bestimmten Zweck erfüllt.

Auch nach Banks (1998) ist ein Modell immer eine Vereinfachung und damit eine Abstraktion eines Systems. Das Modell muss dabei trotzdem komplex genug sein, um seinen Zweck zu erfüllen, also beispielsweise Fragestellungen zum Verhalten des Systems in einer bestimmten Situation zu beantworten. Dazu muss das Verhalten dieses Systems natürlich durch das Modell ausreichend genau simuliert werden können. Trotzdem soll das Modell noch einfach genug sein, um dieses Verhalten zu verstehen und die Simulation effizient zu halten. Anders

Abbildung 2.1: Original und Modell einer Brücke: Auf der linken Seite ist das reale Objekt[1] abgebildet, rechts die Repräsentation durch ein abstrahiertes Modell.
Quelle: Autobahn GmbH des Bundes, Niederlassung Südbayern.

ausgedrückt sollte ein Modell entsprechend seines Zwecks nur so komplex wie nötig und dabei so einfach wie möglich sein.

Nach Benyon (1990) lautet die Definition eines Modells wie folgt: *„A model is a representation of something, constructed and used for a particular purpose.*[2] *“* Dabei wird der Prozess der Modellerstellung als Modellierung (*modeling*) bezeichnet und der Ersteller des Modells als Modellierer (*modeler*). Ein Modell wird durch einen Nutzer (*interpreter*) interpretiert und genutzt. Sowohl für den Modellierer als auch für den Nutzer hat ein Modell einen bestimmten Zweck (*purpose*). Wichtig ist vor allem, dass ein Modell nur dann eine Bedeutung hat, wenn es in einem bestimmten Kontext und für einen bestimmten Zweck eingesetzt wird (pragmatisches Merkmal nach Stachowiak (1973)). Dies ist in Abbildung 2.2 dargestellt.

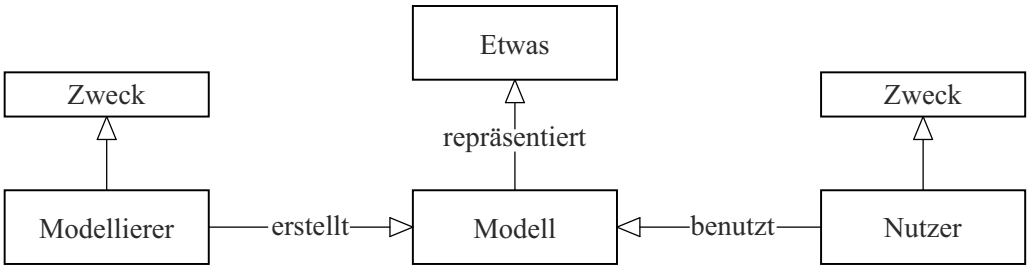


Abbildung 2.2: Die pragmatische Nutzung eines Modells nach Stachowiak (1973) und Benyon (1990). *Eigene Darstellung angelehnt an Overbeek, 2006.*

Diese allgemeinen Aussagen über Modelle spiegeln sich auch in den Modellen wider, die für den Entwurf von Gebäuden und Bauwerken eingesetzt werden. Der Modellbegriff umfasst dementsprechend die verschiedenen abstrahierten Möglichkeiten zum Austausch von Informationen im Kontext des Bauwesens. Eine zweidimensionale Zeichnung ist in diesem Sinne dem Modellbegriff genauso zugeordnet, wie ein dreidimensionales digitales Modell, das sowohl Informationen zur Geometrie und zur Semantik eines Objekts enthält.

[1] Im eigentlichen Sinne des Begriffs ist natürlich auch die Fotografie der Brücke ein Modell.
[2] Ein Modell ist eine Repräsentation von etwas, die für einen bestimmten Zweck konstruiert und verwendet wird. Benyon, 1990, s. 49.

Die pragmatische Nutzung eines Modells aus Abbildung 2.2 wird in Abbildung 2.3 in Anlehnung an Overbeek (2006) durch ein konkretes Beispiel aus dem Bauwesen verdeutlicht: Der Modellierer ist hier ein Architekt, der ein Gebäude entworfen hat. Damit ein Stockwerk dieses Gebäudes durch eine Baufirma gebaut werden kann, hat er ein Modell dieses Stockwerks in Form eines Grundrisses (als 2D-Zeichnung) erstellt, um seinen Entwurf zu beschreiben.

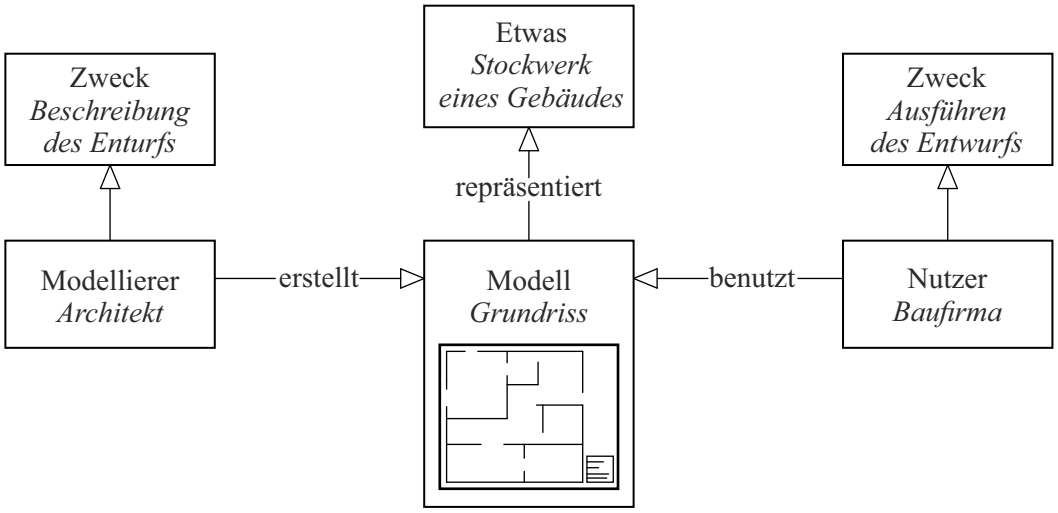


Abbildung 2.3: Beispielhafte Darstellung des pragmatischen Merkmals im Kontext des Bauwesens.

Im hier dargestellten Beispiel ist erkennbar, dass Modellierer und Nutzer das Modell für unterschiedliche Zwecke verwenden. Trotzdem müssen alle Beteiligten das Modell auf die gleiche Art und Weise und mit dem gleichen Ergebnis für ihren jeweiligen Zweck erstellen und interpretieren. Hierfür ist es notwendig, dass gewisse Standards definiert werden, die bei der Erstellung und Interpretation eines Modells als Rahmenbedingungen definiert werden. Im Beispiel in Abbildung 2.3 können dies Konventionen zur Verwendung bestimmter Linienstärken oder Schraffuren sein.

Digitale Informationsmodelle

Dies ist gerade im Hinblick auf digitale Informationsmodelle, die mithilfe entsprechender Softwareanwendungen erstellt, interpretiert und genutzt werden, von wesentlicher Bedeutung. Hier werden digitale Modelle eines Originals in einem Programm modelliert, wobei die entsprechenden Programme in der Regel so aufgebaut sind, dass die Modellierung bestimmten Konventionen folgt. Voraussetzung hierfür ist die Festlegung, wie ein Modell beschaffen sein muss, um diesen Konventionen zu folgen. Da diese Festlegungen eine Metaebene der Modellierung bilden, wird zwischen einem Modell und dem Schema eines Modells unterschieden (Turk, 2001).

Turk (2001) unterscheidet hier zwischen der *konzeptionellen Modellierung (conceptual modelling)* und der *ingenieurtechnischen Modellierung (engineering modelling)*. Ein konzeptionelles Modell ist dabei die Grundlage eines Datenschemas, das übergeordnet spezifiziert, wie einzelne Instanzen eines konkreten Datenmodells (als Resultat der ingenieurtechnischen Modellierung) beschrieben werden. Dieses Datenmodell (auch Produktmodell) beschreibt hingegen ein konkretes Produkt oder den Entwurf eines Produkts, es handelt sich also um eine ausgeprägte Instanz. Auch wenn die Unterscheidung zwischen einem Modell und dem Schema des Modells von enormer Wichtigkeit ist, werden diese Begrifflichkeiten allerdings teils unterschiedlich und nicht immer konsequent verwendet. Wie der folgende Absatz zeigt, ist dies insbesondere bei dem Begriff *Datenmodell* der Fall, sodass eine klare Definition für die in in dieser Arbeit verwendeten Begrifflichkeiten notwendig ist.

Um den Modellbegriff zwischen konzeptionellem Modell und Datenmodell unterscheiden zu können, wurde durch die Object Management Group (OMG) der Begriff Meta Object Facility (MOF) zur Beschreibung einer spezielle Metadaten-Architektur eingeführt (Object Management Group, 2019). Diese standardisierten Spezifikationen der Meta Object Facility unterscheiden zwischen den Instanzdaten (M0), dem Datenmodell (M1), dem Metamodell (M2) und dem Meta-Metamodell (M3). Im Rahmen dieser Arbeit werden diese Bezeichnungen für die verschiedenen Ebenen der Hierarchie der Metamodellierung verwendet, wobei in erster Linie die M0-Instanzdaten und das M1-Datenmodell von Bedeutung sind.

M0-Instanzmodelle können zur Abgrenzung vom generellen Modellbegriff auch als (Bau-)Produktmodelle bezeichnet werden, da sie dazu dienen, ein Produkt in Form eines Gebäudes oder eines Bauwerks zu entwerfen, herzustellen oder zu verwalten und damit der Lage sind, alle wichtigen Aktivitäten während des gesamten Lebenszyklus eines Gebäudes zu unterstützen (Eastman, 1999). Ein M1-Datenmodell für den Austausch solcher Produktmodelle sind beispielsweise die Industry Foundation Classes (siehe Seite 17). Gerade im Hinblick auf die Nutzung eines Produktmodells zur Unterstützung des Planungs- und Herstellungsprozesses ist allerdings anzumerken, dass das (in diesem Fall präskriptive) Modell anfänglich kein real existierendes Original repräsentiert, da das Original zu diesem Zeitpunkt lediglich eine Idee ist – die jedoch mit Unterstützung durch das Modell realisiert werden soll. Ein deskriptives Modell würde hier ein Bauwerk nach seiner Fertigstellung beschreiben.

Es zeigt sich, dass der Modellbegriff je nach Kontext unterschiedliche Bedeutungen haben kann und daher konkret festgehalten werden muss, was mit einem Modell gemeint ist. In diesem Kapitel wird der Begriffe Modell synonym zum Begriff Produktmodell (auf M0-Instanzebene) verwendet, wenn nicht gesondert auf einen Unterschied hingewiesen wird. Dieser Begriff bezeichnet dabei in der Regel ein dreidimensionales digitales Modell, das die Geometrie und die Semantik eines Gebäudes oder eines Bauwerks umfasst. Dies schließt explizit solche Modelle mit ein, die das Resultat eines Modellierungsprozesses sind, der im

Rahmen dieser Arbeit betrachtet wird. Ist ein konzeptionelles Produktmodell (Turk, 2001) gemeint, das sich auf ein Schema zum Austausch und zur Speicherung von Produktmodellen bezieht, so ist dies entsprechend der MOF gekennzeichnet.

2.1.2 Produktmodelle im Bauwesen

Der Modellbegriff kann sich entsprechend der vorigen Definition auf verschiedene Arten des Informationsaustausches im Bauwesen beziehen. Er schließt damit im weiteren Sinne neben den naheliegenden bereits erwähnten zweidimensionalen Zeichnungen zusätzlich beispielsweise Bauteilisten, Leistungsverzeichnisse oder dimensionsreduzierte statische Modelle mit ein. Im allgemeinen Sprachgebrauch versteht man heute allerdings unter einem Modell im Kontext des Bauwesens in der Regel ein dreidimensionales digitales Produktmodell oder Bauwerksmodell, das ein Gebäude oder den Entwurf eines Gebäudes bzw. eines Bauwerks repräsentiert.

Building Information Modeling

Mit der Nutzung dreidimensionaler digitaler Produktmodelle hat sich auch der Begriff des Building Information Modeling (BIM) im Bauwesen etabliert. BIM steht für das Konzept der durchgängigen Nutzung digitaler Bauwerksmodelle für alle Phasen im Lebenszyklus eines Bauwerks.

Standards für die Erstellung, für den elektronischen Austausch sowie zur Speicherung der Daten solcher Produktmodelle, wurden in den vergangen Jahrzehnten bereits in etlichen Industriezweigen entwickelt und erfolgreich verwendet. Die Automobilindustrie erzielte schon vor der Jahrtausendwende Fortschritte hinsichtlich der praktischen Anwendung digitaler Modelle in der Entwicklung und Fertigung von Produkten (Tolman, 1999), sodass inzwischen beispielsweise durch vorgelagerte Simulationen deutliche Effizienzgewinne möglich sind (Heindorf, 2010). Auch wenn im Bauwesen der Einsatz digitaler Produktmodelle schon ähnlich lange angeregt wird (Eastman, 1999; McKinney *et al.*, 1998), ist in der Bauindustrie eine Anwendung in der Breite erst seit dem Anfang des letzten Jahrzehnts im Rahmen der immer weiter fortschreitenden Einführung und Entwicklung des Building Information Modeling (BIM) zu beobachten. Dies zeigt sich sowohl in der zunehmenden praktischen Umsetzung bei Planung und Ausführung von Bauprojekten, als auch durch die Vielzahl von wissenschaftlichen Fachartikeln und Forschungsprojekten. Als Gründe für den im Vergleich zur Automobilindustrie eher späten Paradigmenwechsel hin zu digitalen Produktmodellen nennen Borrmann *et al.* (2021d) die zum Teil schwierigen Randbedingungen im Bauwesen. Beispielhaft ist hier genannt, dass die Prozess- und Wertschöpfungsketten nicht bei einzelnen Unternehmen liegen, sondern sich über viele verschiedene Beteiligte (Architekturbüros, Fachplaner, Baufirmen, Betreiber) verteilen. Neben dieser Fragmentierung der Bauindustrie

sind die Prozesse und Abläufe gerade in der Planungsphase in vielen Fällen auch durch gesetzliche Festlegungen, Richtlinien und Regelwerke festgeschrieben, was zu längeren Innovationszyklen, die zusätzlichem mit einem großen Aufwand verbunden sind, führt.

Die grundlegenden Konzepte des BIM werden unter anderem von (Eastman *et al.*, 2011) und (Borrmann *et al.*, 2021c) ausführlich hinsichtlich der technologischen Grundlagen und der verschiedenen praktischen Anwendungsmöglichkeiten beschrieben. Die wesentliche Grundlage bildet hierbei immer ein strukturiertes, digitales Produktmodell, das die Geometrie und die Semantik eines Entwurfs beinhaltet und als *Building Information Model* bezeichnet wird. Ein solches BIM-Modell soll ein Gebäude oder Bauwerk über seinen gesamten Lebenszyklus hinweg beschreiben.

Der Lebenszyklus schließt dabei die Planung, den Entwurf, den Bau, den Betrieb, die Wartung und Erhaltung sowie gegebenenfalls die Stilllegung, den Abbruch oder den Umbau (Eastman, 1999) mit ein. Das Modell muss hierfür alle relevanten Informationen für alle Projektbeteiligten enthalten. Neben der eigentlichen 3D-Geometrie der einzelnen Bauteile können zusätzlich Informationen zu den verwendeten Materialien (z. B. Stahl oder Beton), deren Eigenschaften (Stahlgüte, Bewehrungsgrad etc.) sowie Mengen-, Kosten- und Prozessinformationen gespeichert werden. Diese Informationen sind der semantische Teil des Modells. Wird ein BIM-Modell mit einem zugehörigen Terminplan kombiniert, spricht man üblicherweise von einem 4D-Modell, wobei die vierte Dimension die Zeit ist. Wird ein Modell durch Informationen zu den zeitlichen Abläufen von Bauvorgängen erweitert, kann es beispielsweise zur Planung oder Kontrolle von Bauprozessen genutzt werden (Braun, 2020; Braun *et al.*, 2014).

Wird ein Gebäude- oder Bauwerksmodell als BIM-Model bezeichnet, lässt sich heute daraus in der Regel ableiten, dass es sich um ein digitales dreidimensionales Modell handelt, das sowohl Geometrie als auch Semantik beinhaltet. Umfang und Tiefe dieser Modelle hinsichtlich geometrischer Genauigkeit, Widerspruchsfreiheit und der Vollständigkeit der enthaltenen Informationen schwanken allerdings erheblich (Preidel, 2020), wobei die Anforderungen an den Informationsgehalt strikter als bei konventionellen Zeichnungen sind (Eastman *et al.*, 2009). Außerdem hängen die Inhalte eines Modells immer davon ab, für welchen Anwendungsfall es verwendet werden soll.

Die Verwendung eines oder mehrerer BIM-Modelle über den gesamten Lebenszyklus eines Bauwerks hinweg eröffnet eine Vielzahl von Möglichkeiten zum Einsatz digitaler Methoden zur Verbesserung der Effizienz und Reduzierung der Fehler und Probleme im Zuge eines Bauprozesses (Franz *et al.*, 2019). Diese Methoden werden als BIM-Anwendungsfälle bezeichnet und stellen den eigentlichen Kern der BIM-Methode dar (Kreider *et al.*, 2013). Das BIM-Modell ist dabei nur das Mittel zum Zweck, mit dem die Umsetzung von sogenannten BIM-Anwendungsfällen überhaupt erst ermöglicht wird, da es die strukturierte Datenbasis bildet, die für die Verwendung digitaler Technologien notwendig ist.

Das Datenmodell Industry Foundation Classes

Auch wenn das Bauwesen erst im Zuge der Entwicklung von BIM damit begonnen hat, die konventionellen zeichnungsbasierten Planungsmethoden durch den Einsatz von Produktmodellen zu unterstützen, wurden neben den bereits genannten Arbeiten von Eastman (1999) schon in den 1990er-Jahren in verschiedenen Forschungsprojekten und im DFG-Schwerpunktprogramm „Objektorientierte Modellierung in Planung und Konstruktion“ (Hartmann, 2000) Methoden erarbeitet, die die Grundlage für den Einsatz von Produktmodellen im Bauwesen gelegt haben. Ein wesentlicher Anspruch an solche Produktmodelle ist dabei der verlustfreie interdisziplinäre Datenaustausch in allen Phasen von konzeptioneller Planung über die Ausführung hin zum Betrieb eines Gebäudes. Die Grundlage bildet eine Modellierung nach dem objektorientierten Paradigma, die sowohl dreidimensionale Geometrie als auch Semantik beinhaltet (Eastman, 1999).

Eine wesentliche Herausforderung, die im Zuge der Nutzung von Produktmodellen im Bauwesen bzw. im Zuge der Verbreitung von BIM auftritt, besteht darin, dass die erstellten digitalen Modelle zwischen den verschiedenen Parteien, die am Planungsprozess beteiligt sind, ausgetauscht werden müssen. Aufgrund der Vielzahl an verfügbaren Software-Produkten, die sowohl zur Modellierung (sog. BIM-Autorenwerkzeuge) als auch für die Analyse, Weiterverarbeitung- und Nutzung der Modelle verwendet werden, ist dafür ein einheitliches standardisiertes Datenformat bzw. -schema notwendig. Dies betrifft weniger den Austausch von Modellen zwischen den Autorenwerkzeugen als die Übergabe der Modelle in nachgelagerte Programme, z. B. zur Mengenermittlung oder Modellprüfung. Die als ISO-Standard 16739 vorliegenden Industry Foundation Classes (IFC) haben sich für diesen Zweck durchgesetzt und werden inzwischen auch durch einen Großteil der verfügbaren Software-Anwendungen auf dem Markt zumindest teilweise unterstützt.

Das herstellerneutrale IFC-Format wird seit 1994 entwickelt (Laakso *et al.*, 2012), damit hochwertige geometrisch-semantische Daten zwischen verschiedenen Software-Anwendungen ausgetauscht werden können. Die IFC werden von der internationalen Non-Profit-Organisation buildingSMART entwickelt und gepflegt und ermöglichen die Interoperabilität zwischen BIM-Softwareanwendungen verschiedener Softwarehersteller (Borrmann *et al.*, 2021a; buildingSMART, 2017). Da dieses Datenformat quelloffen in und durch ein öffentlich einsehbares Schema definiert ist (mittels der Modellierungssprache EXPRESS, definiert nach ISO 10303), können die in einem IFC-Modell gespeicherten Daten generell auch ohne eine bestimmte proprietäre Software gelesen werden. Durch den großen Umfang der IFC in Kombination mit der hohen Genauigkeit bei der Repräsentation ist es allerdings teilweise eine Herausforderung, bestimmte Informationen aus einem IFC-Modell abzufragen (Hartmann *et al.*, 2017). Diese Problematik wurde nach Hartmann *et al.* (2017) allerdings in verschiedenen Forschungsprojekten adressiert (Borrmann *et al.*, 2009; Daum *et al.*, 2014; Langenhan *et al.*, 2013). Der große Umfang der IFC stellt auch die Softwarehersteller vor

große Herausforderungen, da eine vollständige Implementierung des Schemas mit einem erheblichen Aufwand verbunden ist, der sich durch die regelmäßige Überarbeitung und Erweiterung des Schemas weiter erhöht (Amor, 2015). Dies führt in der Regel dazu, dass nur Teile des Schemas in einer bestimmten Softwareanwendung implementiert werden (Hartmann *et al.*, 2017), was bei der praktischen Anwendung zwangsläufig zu Problemen führen kann. Nach Amor (2015) wurde durch BuildingSMART allerdings mit der Einführung von IFC 4 der Versuch unternommen, die Komplexität des Schemas zu reduzieren und damit die Implementierung für Softwarehersteller zu vereinfachen.

In den IFC wird jedes Objekt in einem Modell (ein sog. Modellelement) durch eine Entität dargestellt, die in Beziehung zu einer anderen Entität gesetzt wird. Das Schema basiert auf objektivierten Beziehungen und jede Entität und jede Beziehung wird durch ein eigenes Objekt entsprechend des Schemas definiert. Dadurch lassen sich alle notwendigen Eigenschaften für jedes Modellelement speichern, da zusätzliche Eigenschaften jederzeit hinzugefügt werden können.

Dies gilt genauso für die Geometrie der Modellelemente, die in einem IFC-Modell auf verschiedene Arten repräsentiert werden kann (Borrmann *et al.*, 2021a). Das IFC-Format unterstützt eine Vielzahl der Geometrierepräsentationen die in Kapitel 3 vorgestellt werden. So können u. a. Punkte, Vektoren und Richtungen genauso wie Kurven in 2D und 3D gespeichert werden. Körper können sowohl explizit über ihre Begrenzungen beschrieben werden, als auch implizit durch das Verfahren der Constructive Solid Geometry (CSG) oder durch Rotations-, Extrusions- und Sweep-Körper. Für einfache Körper wie beispielsweise einen aus einem Rechteck extrudierten Quader kann damit sogar eine sehr einfache Konstruktionshistorie der Geometrie eines Modellelements in einem IFC-Modell gespeichert werden. Komplexe assoziativ-parametrische Modelle können mit den IFC allerdings bisher nicht ausgetauscht werden (Borrmann *et al.*, 2018; Ignatova *et al.*, 2015; Theiler *et al.*, 2018), obwohl bereits Untersuchungen in dieser Hinsicht beispielsweise im Bereich des Brückenentwurfs vorgenommen wurden (Girardet *et al.*, 2021; Ji, 2014; Ji *et al.*, 2013).

Aktuell wird das IFC-Datenmodell vornehmlich für die Modellierung von Gebäuden genutzt. Die Entwicklungen hinsichtlich einer Erweiterung für Bauwerke im Infrastrukturbereich wird kontinuierlich vorangetrieben (Amann *et al.*, 2013; Borrmann *et al.*, 2021a) auch wenn sich hier viele Herausforderungen ergeben (Bradley *et al.*, 2016). Obwohl es beispielsweise seit mehreren Jahren Bestrebungen zur Standardisierung im Bereich der Brückenmodellierung (Yabuki *et al.*, 2006) und einzelne Veröffentlichungen im Bereich des Tunnelbaus (Yabuki *et al.*, 2013) gab, wurden die entsprechenden Erweiterungen des IFC-Schemas erst in den letzten Jahren verstärkt angestoßen bzw. vorgenommen. Im Rahmen der Forschergruppe 3DTracks wurde aufbauend auf Vorarbeiten von (Yabuki *et al.*, 2013) ein prototypisches Produktmodell für Tunnel entwickelt, das auch eine mehrskalige Modellierung vorsieht (Borrmann *et al.*, 2013a; Jubierre, 2016; Vilgertshofer *et al.*, 2016b). Weitere Untersuchungen

in dieser Hinsicht wurden von Ninić *et al.* (2020) vorgenommen. Die Erweiterung der IFC für den Infrastrukturbereich wird durch den Infrastructure Room von buildingSMART kontinuierlich vorangetrieben, sodass mit der Version IFC4.3 RC2 (Candidate Status) zum Zeitpunkt der Drucklegung ein Vorschlag vorliegt, der den Austausch von Straßen-, Brücken- und Bahntrassenmodellen ermöglichen soll (BuildingSMART, 2020).

Digitale Zwillinge

Im Kontext von Produktmodellen im Bauwesen gewinnt der Begriff des Digitalen Zwillings[3] (DT) in den letzten Jahren immer mehr an Bedeutung (Brilakis *et al.*, 2020; Lu *et al.*, 2020a). Im Rahmen einer systematischen Literaturanalyse stellen Jones *et al.* (2020) allerdings fest, dass hier eine Vielzahl unterschiedlicher Definitionen vorliegt.

Typischerweise versteht man unter einem Digitalen Zwilling die virtuellen Replica eines physischen Objekts oder Systems. Bei diesem virtuellen Gegenstück handelt es sich letztlich um ein digitales Modell. Dabei kann aber auch das physische Objekt selbst und eine Datenverbindungen zwischen virtuellem Gegenstück und diesem Objekt Teil des Digitalen Zwillings sein (Grieves, 2014). Insbesondere ermöglicht es eine solche Datenverbindung, das digitale Modell kontinuierlich zu aktualisieren, sodass das digitale Abbild mit der physischen Realität übereinstimmt und übereinstimmend gehalten wird. Die dadurch gegebene Aktualität der digitalen Repräsentation stellt den Kernaspekt dieser Technologie dar. Ermöglicht wird dies beispielsweise durch Sensoren am physischen Objekt (El Saddik, 2018), wobei ein Abgleich natürlich auch durch eine Auswertung von Kameradaten oder durch eine manuelle Überprüfung erfolgen kann. Die Häufigkeit dieser Aktualisierungen ist jedoch abhängig von der Art des Produkts, seiner Dynamik, sowie vom Zweck des Modells. Der Digitale Zwilling eines Objekts, das sich kontinuierlich auf relevante Weise verändert (z. B. eines Fahrzeugantriebs), muss demnach häufiger angepasst werden als das eines vergleichsweise statischen Objekts (z. B. eines Bestandsbauwerks).

Grieves *et al.* (2017) unterscheiden weiterhin zwischen zwei Typen Digitaler Zwillinge, den *Digital Twin Prototypes* und *Digital Twin Instances*. Ein DT-Prototyp dient zur Beschreibung prototypischer, also sich noch in Planung befindlicher, physischer Objekte. Es handelt sich dabei also um ein präskriptives Modell, das die Informationen enthält, die notwendig sind, um das physische Objekt zu beschreiben und herzustellen. Die DT-Instanz beschreibt hingegen ein konkretes physisches Objekt oder Produkt, mit dem ein individueller Digitaler Zwilling während der gesamten Lebensdauer dieses physischen Objekts verbunden bleibt (deskriptives Modell).

Im Bauwesen etabliert sich der Begriff des Digitalen Zwillings momentan im Kontext der Nutzung von BIM für die Instandhaltung und den Betrieb von Bauwerken und Gebäuden (Lu

[3] engl. *Digital Twin*

et al., 2020c). Eine DT-Instanz ist hier also letztlich ein *as-built*[4] oder Wie-gebaut-Modell, das den Zustand eines Bauwerks nach Ende der Ausführungphase beschreibt und kontinuierlich, z. B. bei Beschädigungen oder Veränderungen am Bauwerk, aktualisiert wird. Im Rahmen aktueller Forschungsprojekte wird allerdings auch untersucht, inwiefern ein Digitaler Zwilling schon während der Planungs- und Bauphase genutzt werden kann (BIM2TWIN, 2021). Ein solcher Digitaler Zwilling würde dann beispielsweise auch Informationen zum Bauprozess und zum Status des Bauwerks während der Bauphase abbilden und mit fortschreitendem Bauablauf laufend aktualisiert werden. Damit würde es sich hier letztlich um eine Mischung aus DT-Prototpy und DT-Instanz handeln.

2.1.3 Geometrische Modellierung von Produktmodellen

Um ein modellbasiertes Arbeiten mit BIM-Modellen oder Digitalen Zwillingen zu ermöglichen, müssen die dafür notwendigen Modelle zuerst erstellt und zu einem späteren Zeitpunkt gegebenenfalls auch zwischen den Projektbeteiligten ausgetauscht werden. Beispielsweise wird für die Mehrzahl der BIM-Anwendungsfälle ein BIM-Modell benötigt. Je nach den Anforderungen an die geometrische und semantische Detaillierung dieser Modelle (siehe Abschnitt 2.2), ist die Erstellung in der Regel mit einem nicht unerheblichen manuellen Aufwand verbunden.

Bereits vorliegende semantische Informationen können vergleichsweise einfach in ein Modell eingefügt werden, da letztlich nur numerische oder textuelle Werte in entsprechende Datenfelder eingetragen werden müssen. Die Herausforderung liegt dabei also mehr in der Bestimmung des korrekten Werts einer Bauteileigenschaft als darin, diesen Wert dann in ein Modell einzupflegen. Dahingegen muss der geometrische Teil eines Modells computergestützt modelliert werden. Dabei ist zuerst zu entscheiden, wie die Geometrie eines Modells beschaffen sein soll. Anschließend muss die tatsächliche Modellierung in einem geeigneten Softwarewerkzeug durchgeführt werden. Dies betrifft sowohl die Modellierung neuer Bauwerke als auch die Erstellung von Modellen für Bestandsbauwerke. Während die technischen Grundlagen computergestützter Speicherung und Modellierung geometrischer Modelle in Kapitel 3 beschrieben werden, soll hier ein Überblick über den aktuellen Stand der praktischen Anwendung dieser Techniken im Bauwesen gegeben werden.

Als erster Schritt hin zur Erstellung komplexer geometrischer Modelle im Bauwesen kann die flächendeckende Einführung des Computer Assisted Design (CAD) angesehen werden, das in den späten 1980er Jahren das papierbasierte Zeichnen von Plänen sukzessive abgelöst hat (Björk *et al.*, 2010). Im Zuge der fortschreitenden Entwicklung der CAD-Softwarewerkzeuge erfolgte hier eine immer weitergehende Unterstützung bei der Erstellung von Planzeichnungen, da Änderungen im Vergleich zum papierbasierten Arbeiten einfacher umgesetzt werden

[4] Hier wird auch der Begriff *as-maintained* verwendet, um ein Modell zu bezeichnen, das für den Betrieb eines Bauwerks genutzt wird.

können. Durch die iterative Natur eines Entwurfsprozesses sind allerdings im Normalfall kontinuierliche Änderungen an den erstellten Plänen notwendig. Da diese Änderungen manuell in die bereits erstellten Planunterlagen eingepflegt werden müssen, ist dieser Prozess insbesondere bei komplexen Bauvorhaben mit einem erheblichen Aufwand verbunden. Dies bedingt sich vor allem dadurch, dass eine Änderung an nur einem Bauteil in allen Schnitten, Ansichten und Grundrissen, in denen dieses Bauteil dargestellt ist, propagiert werden muss.

Durch die Einführung des modellbasierten Arbeitens mit BIM können solche Änderungen inzwischen allerdings weitgehend automatisiert durchgeführt werden. In vielen aktuell auf dem Markt verfügbaren CAD-Werkzeugen (die als BIM-Autorenwerkzeuge bezeichnet werden) bildet ein dreidimensionales Modell die *Single Source of Truth* der Entwurfsgeometrie, von der beliebige Schnitte weitgehend automatisiert abgeleitet werden können. Änderungen an einem Bauteil werden damit direkt in die abgeleiteten Pläne übertragen und müssen nicht mehr manuell nachgeführt werden. Dabei ist allerdings einschränkend zu erwähnen, dass es in vielen Fällen noch nicht möglich ist, Pläne vollautomatisiert abzuleiten, die den Anforderungen gängiger Normen (z. B. der RAB-ING[5]) entsprechen.

Weiterhin hat sich inzwischen eine Vielzahl von BIM-Autorenwerkzeugen am Markt etabliert, bei denen Techniken der parametrischen Modellierung mit objektorientierten Modellierungsansätzen kombiniert werden. Mit diesen Werkzeugen wird die manuelle Arbeit der Konstrukteure wesentlich erleichtert, da diese Programme zunehmend detailliertes und komplexes domänenspezifisches Wissen implementieren (Lee *et al.*, 2006b). Durch den Aufbau umfangreicher Objektbibliotheken können vordefinierte Objekte (z. B. Revit Familien, Allplan Smart Parts, Archicad GDL-Objekte) in ein Modell eingefügt werden, sodass gleiche Bauteile nur einmal als Vorlage modelliert werden müssen. Beispielsweise muss die Geometrie eines Fensters damit nicht mehr aufwändig modelliert werden, sondern ist für bestimmte Typen von Fenstern bereits durch das Autorenwerkzeug oder durch den Hersteller vordefiniert. Zusätzlich können diese Objekte durch die Nutzung parametrischer Modellierungstechniken insofern adaptiv gestaltet werden, als dass sie sich bei Änderungen automatisch anpassen. Wird beispielsweise eine Wand mit einem Stockwerk verknüpft, passt sich die Höhe der Wand automatisch an die Höhe des Stockwerks an. Solche adaptiv-parametrischen Funktionen tragen damit wesentlich zur Verringerung des manuellen Arbeitsaufwands bei, auch wenn sie für den Anwender teils gar nicht mehr direkt als solche zu erkennen sind. Der Anwender kann damit die Vorteile der parametrischen Modellierung nutzen, ohne sich direkt mit der Definition der grundlegenden parametrischen Zusammenhänge auseinandersetzen zu müssen.

Demgegenüber stehen meist aus dem Bereich des Maschinenbaus stammende parametrische CAD-Anwendungen, die als leistungsstarke Hilfsmittel in Konstruktionsprozessen dienen, indem sie einen flexiblen Ansatz für die Erzeugung einer großen Anzahl von Konstruktions-

[5]Richtlinien für das Aufstellen von Bauwerksentwürfen für Ingenieurbauten.

alternativen bieten (Jubierre, 2016; Kondyli *et al.*, 2018). Hierbei steht die Definition der parametrischen Zusammenhänge in einem Modell im Mittelpunkt des Modellierungsvorgangs und kann durch den Nutzer vollumfänglich innerhalb des Funktionsumfangs der jeweiligen Anwendung gesteuert werden. Um eine Nachvollziehbarkeit der Parametrik zu gewährleisten, muss allerdings während des Entwurfs- und Modellierungsprozesses festgehalten werden, wie das parametrische Modell aufgebaut wird (Lee *et al.*, 2006b). Somit lassen sich komplexe parametrische Geometrien für einen Entwurf erstellen, die durch die Änderung einzelner Parameterwerte an veränderte Randbedingungen angepasst werden können.

Im Vergleich zu den zuvor beschriebenen BIM-Autorenwerkzeugen bieten parametrische CAD-Anwendungen eine größere Flexibilität hinsichtlich der erstellbaren Geometrie und der durch die parametrischen Zusammenhänge definierten Topologie eines Modells. Durch die größere Flexibilität steigt allerdings einerseits der anfängliche Modellierungsaufwand und zusätzlich die Anforderungen an die Kenntnisse des Nutzers. Da immer mehr BIM-Autorenwerkzeuge auch grundlegende parametrische Funktionen zur Erstellung wiederverwendbarer Objekte bieten, ist allerdings davon auszugehen, dass in naher Zukunft auch mit diesen Programmen komplexe parametrische Zusammenhänge abgebildet werden können. Dies setzt jedoch voraus, dass Anwender die nötigen Kenntnisse besitzen, um diese Möglichkeiten zu nutzen. Hier müssen letztlich die Hersteller der CAD-Werkzeuge einen sinnvollen Kompromiss in ihren Programmen finden, der eine hohe Flexibilität für erfahrene Anwender gewährleistet, ohne das Erlernen des Programms unnötig zu erschweren.

Bei der Verwendung von parametrischen CAD-Anwendungen ist aber genauso wie bei BIM-Autorenwerkzeugen festzuhalten, dass es kaum Möglichkeiten gibt, die erstellten Modelle inklusive des über die parametrischen Zusammenhänge abgebildeten Domänenwissens von einem Programm verlustfrei in das Programm eines anderen Herstellers zu übertragen. Auf STEP (ISO 10303) basierende Datenformate wie die IFC können zwar die finale Geometrie eines Modells transportieren, die hintergründige Entwurfsabsicht geht aber verloren (Safdar *et al.*, 2020).

Auch wenn es bereits seit über 20 Jahren Bestrebungen gibt, diese Einschränkung durch eine Erweiterung von STEP (Pratt, 1997, 1998, 2004) zumindest insofern zu beheben, als dass die Konstruktionshistorie parametrischer Modelle herstellerneutral ausgetauscht werden kann, ist bisher keine zufriedenstellende Lösung bekannt. Zwar sind mit *STEP Part 111 - Elements for the procedural modelling of solid shapes* und *STEP Part 242: Application protocol: Managed model-based 3D engineering* Erweiterungen der ISO 10303 konzipiert worden, die auch den Austausch prozeduraler parametrischer Modelle ermöglichen sollen, allerdings sind nach Kim *et al.* (2007) *„die APIs kommerzieller CAD-Systeme nicht vordergründig als Schnittstelle für den Austausch vollständig parametrischer Modelle ausgelegt“*[6]. Den

[6] *„The APIs of commercial CAD systems are not primarily intended as an interface for the exchange of full parametric models.“*

Autoren zufolge kann insofern nicht sichergestellt werden, dass ein parametrisches Modell wirklich vollständig zwischen zwei CAD-Systemen ausgetauscht werden kann. Betrachtet man die gängigen und in der Breite verwendeten parametrischen CAD-Systeme bestätigt sich diese Aussage auch zum Zeitpunkt der Erstellung dieser Arbeit[7]. Diese Beobachtung wird beispielsweise auch in Schätzle (2016) bestätigt. Die Umsetzung von BIM beruht jedoch auch wesentlich auf der Verfügbarkeit geeigneter Tools für den effizienten Austausch von Informationen (Migilinskas *et al.*, 2013).

Problematisch ist in diesem Kontext vor allem, dass sich Planer, Ingenieure und Konstrukteure dauerhaft von den Produkten eines Softwareherstellers abhängig machen. Da dies aber nur durch die konsequente Nutzung proprietärer Software eines bestimmten Herstellers möglich ist, entstünde durch einen Wechsel der verwendeten Software ein erheblicher Aufwand, da bereits vorhandene Objektbibliotheken neu aufgebaut werden müssen.

Dies gilt insbesondere für Objekte, die sehr spezifische Bauteile beschreiben und daher nicht in den Standard-Objektbibliotheken der gängigen Autorenwerkzeuge enthalten sind. Baut also ein Planungsbüro eine interne Bibliothek an komplexen Bauteilobjekten auf, die über parametrische Funktionalitäten bestimmte Modellierungstätigkeiten erleichtern und so die Erstellung von Entwürfen beschleunigen und damit letztlich Kosten senken, ist es an die verwendete Software gebunden.

Auch wenn sich nun der Wechsel zu einer Software eines anderen Herstellers aus finanziellen und technologischen Gründen grundsätzlich anbieten würde, müsste hierbei immer auch bedacht werden, dass die Umstellung Auswirkungen auf bestehende Objektbibliotheken haben wird. Neben dem ohnehin bei der Einführung einer neuen Software anfallenden Schulungsaufwand müssen bestehende Objekte in die neue Software übertragen werden. Dies gilt genauso für Planer, die aufgrund der Vorgaben eines Auftraggebers eine bestimmte Software einsetzen müssen.

Da ein einfacher und weitgehend automatisierter Ex- und Import wie bereits beschrieben kaum möglich ist, fällt potenziell ein erheblicher manueller Aufwand an. Dabei ist es sogar denkbar, dass komplexe Modelle nicht nur auf Basis bestehender Objekte neu erstellt werden müssen, sondern gegebenenfalls eine komplette Neukonzeption notwendig wird. Dies kann beispielsweise dann der Fall sein, wenn die Person, die ein Objekt[8] ursprünglich erstellt hat, nicht mehr verfügbar ist. Das Wissen, das diese Person in die ursprüngliche Erstellung eines Bauteilobjekts eingebracht hat, muss dann neu aufgebaut werden – gerade dann, wenn ein Objekt so komplex aufgebaut ist, dass die inneren Zusammenhänge für

[7] Exportiert man beispielsweise ein parametrisch in Autodesk Inventor 2021 erstelltes Bauteil in eine STEP-Datei und importiert diese Datei anschließend im selben Programm, so ist zwar die Geometrie vollständig vorhanden, die Konstruktionshistorie geht jedoch verloren. Dies ist auch bei Verwendung des genannten Anwendungsprotokolls 242 der Fall.

[8] Mit Objekt ist hier sowohl ein Objekt zur Repräsentation eines einzelnen Bauteils als auch ein komplettes Modell gemeint.

einen anderen Nutzer nicht oder nur mit sehr großem Aufwand nachvollziehbar sind. Dies wird natürlich insbesondere dann relevant, wenn in sich komplexe parametrische Objekte miteinander kombiniert werden, um komplette Bauwerksmodelle zu parametrisieren. Beispielhaft können hier Tunnelbauwerke genannt werden, bei denen die einzelnen Tübbinge als parametrische Objekte erstellt werden und anschließend mittels parametrisch-assoziativer Modellierungsmethoden (siehe Kapitel 3) zu einem parametrisch von der Trassierung abhängigen Gesamtmodell zusammengefügt werden (Jubierre *et al.*, 2015), siehe Abbildung 2.4. Ein ähnlicher Ansatz wurde auch von Luo *et al.* (2021) ausgearbeitet.

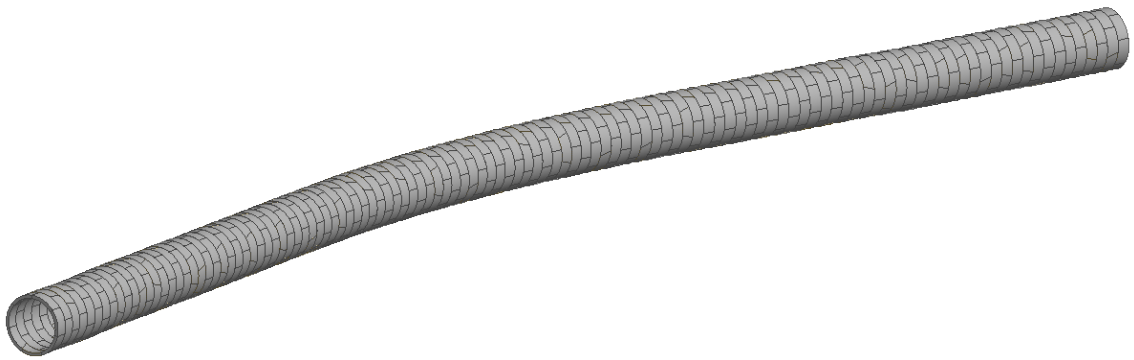


Abbildung 2.4: Beispiel eines parametrisch erstellten Tunnelabschnitts, bei dem Ringe aus Tübbing-Objekten zu einem Gesamtmodell kombiniert wurden. *Quelle: Jubierre, 2016.*

2.1.4 Einordnung

Im Abschnitt 2.1 wurde nach einer allgemeinen Einführung des Modellbegriffs ein Überblick über die Nutzung, den Austausch und die Erstellung von Produktmodellen im Bauwesen gegeben. Zusammenfassend lässt sich feststellen, dass sich durch die Einführung von BIM und die immer weiter verbreitete Nutzung digitaler Modelle eine Vielzahl an Möglichkeiten zur Unterstützung von Entwurfs- und Bauprozessen ergibt.

Um diese Möglichkeiten vollumfänglich nutzen zu können, müssen die entsprechenden Modelle allerdings auch konstruiert werden. Hier hat sich gerade im Hinblick auf die Modellierung der Geometrie in den vergangenen Jahren eine Vielzahl von unterstützenden Softwarewerkzeugen etabliert, die den manuellen Arbeitsaufwand reduzieren. Durch die proprietäre Natur dieser Werkzeuge entsteht allerdings eine Abhängigkeit, da ein Wechsel des Softwarewerkzeugs mit viel Aufwand verbunden sein kann. Hieraus lässt sich die Frage ableiten, wie das in einem Modellierungsprozess enthaltene Domänenwissen herstellerneutral formal gespeichert und ohne die Nutzung eines bestimmten Softwarewerkzeugs abgerufen werden kann.

2.2 Detaillierung von Modellen

Ein weiterer wesentlicher Aspekt, der bei der Erstellung von Modellen berücksichtigt werden muss, ist, dass sie im Rahmen von kontinuierlichen Entwurfs- und Planungsprozessen erstellt werden. Zu Beginn eines Entwurfsprozesses sind die vorliegenden Informationen vergleichsweise vage und werden im weiteren Verlauf schrittweise ausgearbeitet und detailliert. Dies gilt übergreifend für die Bestimmtheit, Genauigkeit, Detailliertheit und Zuverlässigkeit der Informationen in einem Modell (Abualdenien *et al.*, 2021). Insofern soll hier vorgestellt werden, wie der Ausarbeitungsgrad eines BIM-Modells im Zuge des Entwurfsprozesses verfeinert wird. Auch wenn im Rahmen dieser Arbeit die Geometrie des Modells im Vordergrund steht, wird auch auf den semantischen Teil des Modells eingegangen. Zusätzlich wird das Konzept der mehrskaligen Modellierung eingeführt und hinsichtlich der Anwendung im Kontext der Ausarbeitungsgrade von BIM-Modellen eingeordnet.

Entsprechend des Modellbegriffs ist ein Modell eine Abstraktion der Realität (siehe Abschnitt 2.1.1). Wird dasselbe reale Objekt durch unterschiedlich detaillierte Modelle repräsentiert, gilt dementsprechend, dass dieses Modell das Objekt unterschiedlich abstrahiert darstellt. Wird das reale Objekt sehr stark abstrahiert, entsteht ein vergleichsweise grobes Modell, da Details durch die Abstraktion verloren gehen und nicht abgebildet werden. Je weniger stark das Objekt abstrahiert wird, desto mehr Details und Feinheiten bleiben erhalten, es entsteht also ein vergleichsweise konkretes Modell. Der Detaillierungsgrad eines Modells kann auch mit der Komplexität eines Modells verglichen werden und trifft damit eine Aussage darüber, bis zu welchem Grad die Elemente und inneren Zusammenhänge eines Objekts oder eines Systems in einem Modell enthalten sind (Brooks *et al.*, 1996). Allerdings sind die Grenzen zwischen den verschiedenen Stufen der Abstraktion fließend und müssen genau definiert werden, bevor man von bestimmten Abstraktions- oder Detaillierungsgraden sprechen kann.

Die Genauigkeit der in einem bestimmten Abstraktionsgrad dargestellten Informationen entspricht dabei den Anforderungen der beabsichtigten Anwendung. Als typisches Beispiel ist hier die Modellierung von Straßen in kartografischen Anwendungen wie Google Maps oder OpenStreetMap zu nennen. Dort werden beispielsweise Straßen in einem kleinen Kartenmaßstab stark vereinfacht als rein linienförmige Objekte dargestellt, während sie dem Nutzer bei großen Maßstäben als flächige Objekte, durch die auch einzelne Fahrspuren erkennbar sind, angezeigt werden. Ohne die den verschiedenen Maßstäben zugrunde liegenden Abstraktionsgrade wäre eine sinnvolle Nutzung kaum möglich. Dieses Prinzip kann analog auf eine Vielzahl anderer Objekte, die in diesen Anwendungen visualisiert werden, übertragen werden.

2.2.1 Level of Detail

Um Modelle in unterschiedlichen Detaillierungsgraden zu beschreiben, wird in den Bereichen der Computergraphik und der Geographischen Informationssysteme der Begriff Level of Detail verwendet. Da die Definition der Ausarbeitungsgrade von BIM-Modellen anfänglich auch über diesen Begriff erfolgt ist, soll hier eine kurze Einführung erfolgen. Dabei wird konsequent die Abkürzung LoD zur Beschreibung des Level of Detail verwendet, um den Unterschied zum Level of Development (LOD) als Beschreibung des Ausarbeitungsgrads von BIM-Modellen abzugrenzen. In der Literatur wird allerdings auch der Level of Detail oft mit LOD abgekürzt. In dieser Arbeit wird außer im Bereich der Computergrafik die Verwendung der Abkürzung LoD im Plural als LoDs geschrieben, da dies auch in der Literatur weitgehend gängig ist.

Computergrafik

Im Bereich der Computergrafik geben Luebke *et al.* (2003) einen umfangreichen Einblick in die Verwendung der LoD, wobei erste Ansätze in dieser Hinsicht bereits unter anderem von (Clark, 1976) entwickelt wurden. Die Notwendigkeit der Verwendung von verschiedenen *Levels of Detail* ergibt sich aus dem Kompromiss, der bei der computergestützten interaktiven Visualisierung einer virtuellen Welt zwischen Komplexität und Performanz gefunden werden muss. Aufgrund der durch die verfügbare Hardware vorgegebenen Beschränkung der Berechnungsgeschwindigkeit muss dabei zwischen dem Realismus, also der Wiedergabetreue einer visualisierten Szene und einer flüssigen Darstellung mit ausreichender Bildfrequenz abgewogen werden. Die LoD als Themenbereich der Computergrafik versuchen einen möglichst optimalen Kompromiss zwischen Komplexität und Performanz zu erreichen, indem der Detaillierungsgrad einer virtuellen Welt reguliert wird (Luebke *et al.*, 2003). Dabei kommt das Konzept der LoD in verschiedenen Ansätzen zum Einsatz, die letztlich alle die Verbesserung der grafischen Darstellung einer virtuellen Welt ohne Einbußen bei der Performanz zum Ziel haben (Aliaga *et al.*, 1999a; Aliaga *et al.*, 1999b; Hoppe, 1997; Maciel *et al.*, 1995).

Auch wenn umfangreiche Literatur zu diesem Thema vorhanden und die zu lösende Problematik bis heute relevant ist, kann das grundlegende Konzept der LoD vergleichsweise einfach zusammengefasst werden. Die wesentliche Idee besteht darin, dass Objekte, die in einer Szene klein, unwichtig oder weit vom Betrachter entfernt sind, durch ein stark abstrahiertes geometrisches Modell – also wenig detailliert – dargestellt werden (Abbildung 2.5). Die dazu nötigen Stufen der Abstraktion sind die Level of Detail eines Objekts, wobei sich der Grad der Abstraktion bei einer rein geometrischen Betrachtung allein über die Anzahl der Dreiecke definiert. Da in einer Szene in der Regel wenige Objekte sehr nah am Betrachter sind und (zumindest bei einem freien Sichtfeld) deutlich mehr Objekte

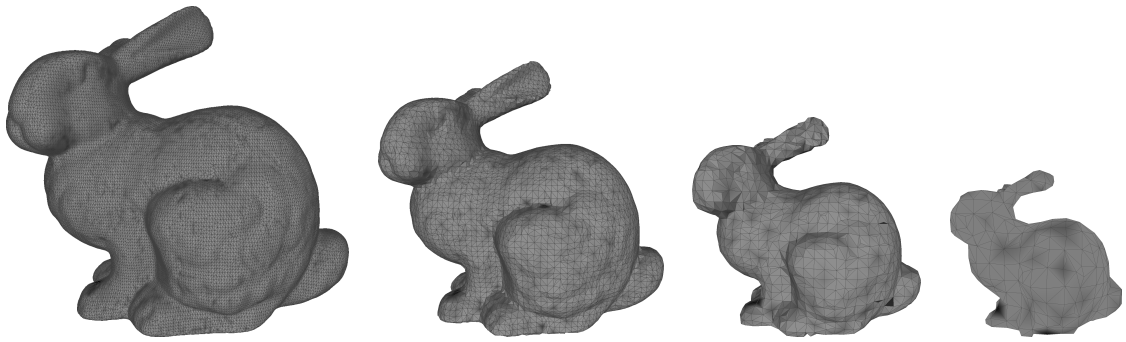


Abbildung 2.5: Das fundamentale Konzept der Level of Detail. Das *Standford Bunny* wird von links nach rechts mit einer jeweils verringerten Anzahl an Dreiecken abgebildet, da es bei einer kleiner werdenden Abbildung weniger genau repräsentiert werden muss. *Eigene Darstellung angelehnt an Luebke* et al.*, 2003.*

weiter entfernt liegen, kann somit die Anzahl der Dreiecke insgesamt deutlich reduziert werden. Da beim Rendering die Anzahl der Dreiecke einen wesentlichen Einfluss auf die zur Erstellung eines Bildes notwendige Berechnungszeit hat (Evans *et al.*, 1996), führt eine Verringerung der Dreiecksanzahl z. B. bei einer interaktiven Visualisierung zu einer Erhöhung der Bildfrequenz und damit zu einer Verbesserung der Nutzererfahrung.

Nach Heok *et al.* (2004) kann das Feld der LoD in vier Bereiche aufgeteilt werden, die auch bei Luebke *et al.* (2003) Erwähnung finden und aufeinander aufbauen. *Diskrete LoD* werden vor dem eigentlichen Rendern für jedes Objekt separat durch die stufenweise Vereinfachung des Ausgangsobjekts erstellt. Während der Laufzeit eines Programms wird dann auf Basis der o. g. Kriterien einer dieser LoD des Objekts ausgewählt. Da der Erstellungsprozess der einzelnen LoD nicht während der Laufzeit stattfindet, hat er keinen negativen Einfluss auf die Rendergeschwindigkeit. Werden *kontinuierliche LoD* verwendet, so wird der für eine Szene gewünschte LoD eines Objekts während der Programmausführung erzeugt. Zur Erzeugung wird auf eine im Vorfeld erstellte Datenstruktur, die eine kontinuierliche Detaillierung beinhaltet, zurückgegriffen. Der Vorteil liegt hier in der Granularität, da für eine Szene ein LoD gewählt wird, der für die visualisierten Objekte nicht mehr Dreiecke als notwendig vorsieht.

Dieses Vorgehen kann durch *View-Dependent LoD* erweitert werden. Hier wird der optimale LoD in Abhängigkeit vom Blickfeld dynamisch bestimmt, wobei ein Objekt auch in mehreren Detailstufen vorliegen kann. Bei einem großen Objekt wird der Teil, der nahe beim Betrachter liegt, detaillierter dargestellt als ein weiter entfernter Teil. Mit dieser Methode kann auch sichergestellt werden, dass der Rand eines Objektes immer ausreichend detailliert dargestellt wird. Damit wird die Qualität der Visualisierung weiter verbessert. Dieses Vorgehen ist insbesondere bei großen oder schlecht segmentierten Objekten hilfreich, wie sie z. B. durch den Scan eines realen Objekts entstehen können. Durch die Verwendung von *Hierarchischen*

LoD können vergleichsweise kleine Objekte gruppiert und gesammelt berechnet werden, sodass Performanzeinbußen durch eine große Anzahl kleiner Objekte vermieden werden.

Zusammenfassend lässt sich feststellen, dass der LoD im Kontext der Computergrafik ein Modellelement rein aus Sicht der Visualisierung beschreibt. Niedrigere LoD werden durch Abstraktion des höchsten LoD erstellt. Den Ausgangspunkt bildet also immer ein Modell, dass so weit wie nötig (oder wie möglich) dem abzubildenden realen oder gedachten Objekt entspricht und damit den höchsten LoD darstellt. Durch die stufenweise oder kontinuierliche Abstraktion dieses Modells entstehen anschließend die niedrigeren LoD. Damit hängen die gröberen LoD zwangsläufig von dem feinsten LoD ab.

Vermessung, Kartographie und 3D-Stadtmodelle

Relevant für den in dieser Arbeit verfolgten Ansatz ist das in CityGML verankerte Konzept der mehrskaligen Modellierung und die damit einhergehende Abbildung von verschiedenen LoDs, die in CityGML zum Einsatz kommen. Hintergrund ist hier das in der Kartographie schon lange etablierte Konzept, Karten in verschiedenen Maßstäben darzustellen, um für verschiedene Anwendungsszenarien einen sinnvollen Kompromiss aus Übersichtlichkeit und Detailtiefe zu ermöglichen. Sester (2005) stellt fest, dass die damit einhergehende Abstraktion und Generalisierung für die Nutzung digitaler räumlicher Datenbestände mindestens genauso wichtig wie für analoge Datenbestände ist. Zur Beschreibung der 3D-Stadtmodelle werden daher in CityGML fünf verschiedene LoD unterschieden (siehe Abbildung 2.6). Durch CityGML werden standardisierte Vorgaben für die geometrische und semantische Repräsentation von realen Objekten in den verschiedenen LoDs gemacht. Dabei werden im LoD 0 nur sehr wenige Details betrachtet, während in LoD 4 Objekte mit einer sehr großen Detailtiefe modelliert werden können. Die geometrische Detailtiefe und die semantische Komplexität nehmen also mit jedem LoD zu, das heißt niedrigere LoDs haben einen höheren Grad der Abstraktion.

Bei CityGML handelt es sich um einen XML-basierten Standard zur Speicherung, Repräsentation und zum Austausch von virtuellen 3D-Stadt- und Landschaftsmodellen (Kolbe, 2009; Kolbe *et al.*, 2005), der aktuell in der Version 2.0 vorliegt. Der Standard erlaubt die Beschreibung dreidimensionaler Stadtmodelle in Bezug auf ihre Geometrie, Topologie und Semantik und ihr Aussehen. Durch das offene und herstellerneutrale Format zur Speicherung und zum interdisziplinären Austausch konnte sich CityGML als internationaler, vom Open Geospatial Consortium (OGC) anerkannter Datenstandard etablieren (Brüggemann *et al.*, 2015). Inzwischen ist CityGML der wichtigste internationale Standard zur dreidimensionalen Modellierung von Städten und Landschaften in 3D mit umfangreicher Semantik (Arroyo Ohori *et al.*, 2018).

CityGML ermöglicht sowohl die reine Visualisierung von 3D-Stadtmodellen, als auch die Abfrage und Auswertung von semantischen Informationen zur weiteren Verwendung und Analyse. Daraus ergibt sich ein breites Feld hinsichtlich der möglichen Nutzung. Biljecki *et al.* (2015) zeigen in einer Studie, dass 3D-Stadtmodelle in über 29 Anwendungsfällen eingesetzt werden können. Auch wenn nicht bei allen dieser Anwendungsfällen CityGML zum Einsatz kommt, zeigt sich dadurch doch die generelle Relevanz von 3D-Stadtmodellen. CityGML kommt hier beispielsweise bei der Abschätzung der Energiebilanz von Wohngebäuden zum Einsatz (Agugiaro, 2016).

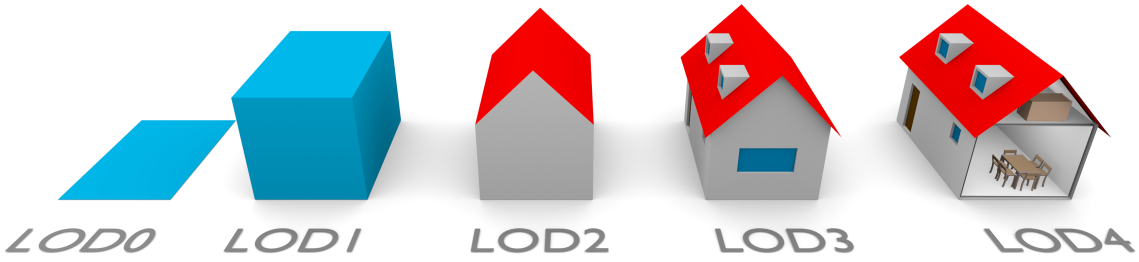


Abbildung 2.6: Die fünf LoDs des CityGML 2.0 Standards. *Quelle: Biljecki, 2017, S. 7.*

Einen ausführlichen Überblick über die genaue Definition der LoDs in CityGML gibt die Dissertation von Biljecki (2017), in der diese auch kritisch diskutiert werden. Biljecki (2017) und Ohori *et al.* (2015) stellt hier fest, dass in der Version 2.0 von CityGML die einzelnen LoDs nicht ausreichend genau spezifiziert sind und außerdem insofern zu generisch, als dass in einem LoD stark unterschiedliche Abstraktionsgrade enthalten sein können. Es gibt inzwischen verschiedene Vorschläge das LoD-Konzept in CityGML zu überarbeiten bzw. zu erweitern. Biljecki *et al.* (2016) schlagen beispielsweise vor, den CityGML Standard insofern zu erweitern, als dass in einem LoD mehr als eine geometrische Repräsentation gespeichert werden kann, um verschiedenen geometrischen Varianten Rechnung zu tragen. Tang *et al.* (2020b) stellen ein neues LoD-Modellierungsparadigma vor, das die Erstellung kontextbezogener 3D-Stadtmodelle ermöglicht, die auf bestimmte Anwendungen zugeschnitten sind.

In CityGML werden die Mehrfachrepräsentationen in den verschiedenen LoDs ohne explizite Beschreibungen von Abhängigkeiten verwendet (Biljecki *et al.*, 2015). Jedes Objekt hat also in jedem Detaillierungsgrad eine eigenständige Repräsentation, die unabhängig von den Repräsentationen des selben Objekts in anderen LoD definiert ist. Dementsprechend ist eine automatische Konsistenzsicherung im Fall von Modifikationen nur sehr eingeschränkt möglich, Änderungen müssen stattdessen manuell in allen LoDs nachgeführt werden. Bei der Beschreibung bereits existierender Bauwerke stellt dies allerdings kein wesentliches Problem dar, da die niedrigeren LoD vom realen Objekt abgeleitet werden können. Zusätzlich ist es auch möglich, niedrigere LoDs durch Vereinfachung von einem höheren LoD abzuleiten (Fan *et al.*, 2012). In der Karthographie spricht man hierbei von *Generalisierung*.

Durch das Vorhandensein allgemeingültiger Definitionen der einzelnen LoDs bildet CityGML im Vergleich zur Computergrafik eine Grundlage für die Beschreibung der Detaillierungsgrade von Produktmodellen im Bauwesen. Es muss daher durchaus festgehalten werden, dass die Definition von Ausarbeitungs- und Detaillierungsgraden im Kontext von Building Information Modeling in Teilen auf diesen Konzepten aufgebaut wurde.

2.2.2 Detaillierungsgrade im Bauwesen

Solange die Planung im Bauwesen rein auf Basis von 2D-Zeichnungen erfolgte, konnte der Detaillierungsgrad, in dem ein Bauteil zu einem bestimmten Zeitpunkt definiert sein musste, weitgehend über den Maßstab der Zeichnung erfolgen. In frühen Planungsphasen sind kleine Maßstäbe ausreichend, während im weiteren Verlauf immer größere Maßstäbe[9] notwendig werden, um eine ausreichend genaue Beschreibung eines Entwurfs zu gewährleisten. Dahingegen kennen BIM-Modelle keine unterschiedlichen Maßstäbe. Die geometrische Modellierung wird sinnvollerweise für das komplette Modell im Maßstab 1:1 erstellt und lediglich die vom Modell abgeleiteten Pläne werden in einem bestimmten Maßstab dargestellt. Bei der Erstellung des Modells können geometrische Details, die am Anfang des Entwurfsprozesses noch nicht ausgearbeitet sind, daher nicht durch die Verwendung eines kleinen Maßstabs „kaschiert“ werden.

Insofern kann ein BIM-Modell bereits nach der ersten und gegebenenfalls rein konzeptionellen Modellierung bereits scheinbar sehr detailliert wirken und rein geometrisch einen hohen geometrischen LoD aufweisen, was zu falschen Annahmen führen kann (Abualdenien *et al.*, 2019). Dieser Effekt verstärkt sich auch durch die Modellierung nach dem objektorientierten Paradigma in den gängigen BIM-Autorenwerkzeugen. Wird beispielsweise ein Fenster aus einer Objektbibliothek in das Modell eingefügt, so ist das so erstellte Modellelement in der Regel bereits sehr detailliert. Dass sich sowohl die Abmessungen als auch die Position und der Typ des Fensters im weiteren Verlauf der Planung noch ändern können, ist nicht direkt erkennbar. Im Zuge der Verbreitung von BIM entstand durch diese Problematik teils eine große Unsicherheit bei Planern und Bauherren hinsichtlich der Anforderungen an die Detailtiefe, die zu einem bestimmten Zeitpunkt in einem Modell enthalten sein soll. Dies gilt gleichermaßen für die geometrischen und für die semantischen Informationen. Diese Unsicherheit begründet sich einerseits in der Frage, welche Informationen bei einer BIM-basierten Planung wann notwendig sind und zusätzlich darin, dass keine allgemeingültige Definition zur Kommunikation der Detailtiefe eines Modells verfügbar war.

Daraus ergab sich die Notwendigkeit, ein Konzept zur Beschreibung der geometrischen und semantischen Informationen zu entwickeln, die während des Entwurfsprozesses von

[9] Die Adjektive *groß* und *klein* beziehen sich auf die Größe eines Objektes auf der Zeichnung. Der Maßstab 1:10, der für eine Detailzeichnung verwendet wird, ist also *größer* als der Maßsstab 1:100, der in der Entwurfsplanung zum Einsatz kommt.

Bauwerken zu einem bestimmten Zeitpunkt vorhanden sein sollen. Aus dieser Fragestellung entstand letztlich das Konzept der Levels of Development (LOD), wobei anfänglich auch im Bauwesen der in der Computergrafik und in der 3D-Stadtmodellierung bereits etablierte Begriff Level of Detail verwendet wurde.

Im Bauwesen hat sich aber letztlich der Begriff Level of Development (LOD) gegenüber LoD durchgesetzt, da er den Ausarbeitungsgrad, also die Reife, Vollständigkeit und Zuverlässigkeit bzw. Vagheit, der geometrischen und semantischen Informationen der Bauteile in einem Modell darstellen kann. Nach anfänglichen Entwicklungen des Konzepts durch das American Institute of Architects (AIA) wurde das LOD-Konzpet schließlich durch die amerikanische Organisation BIMForum weiter ausgearbeitet und wird regelmäßig in einer neuen Version veröffentlicht (BIMForum, 2020). Dieses Konzept umfasst aktuell sechs LODs, die in der Regel als eine Kombination aus dem semantischen und dem geometrischen Ausarbeitungsgrad eines Bauteils verstanden werden. Der geometrische und der semantische Ausarbeitungsgrad eines Bauteils werden dabei als Level of Geometry bzw. Level of Information bezeichnet. Das LOD-Konzept hat sich inzwischen weltweit als quasi-Standard durchgesetzt, auch wenn in verschiedenen Ländern andere Vorschläge erarbeitet wurden (Abualdenien *et al.*, 2021).

Da die LOD entsprechend der Definition des BIMForums sowohl in Deutschland als auch global im Zuge der BIM-Umsetzung immer wichtiger werden und damit auch hinsichtlich des Vorgehens bei der geometrischen Modellierung relevant sind, sollen sie im Folgenden kurz vorgestellt werden. Dabei wird neben der Definition des LOD auch auf die LOG und LOI eingegangen.

Level of Development

Der LOD wird verwendet, um beim Daten- bzw. Informationsaustausch mittels BIM-Modellen den Ausarbeitungs- oder Fertigstellungsgrad der Modellelemente, aus denen ein Modell besteht, zu definieren. Diese Definition ist notwendig, um sicherzustellen, dass zwischen allen Projektbeteiligten ein gemeinsames Verständnis hinsichtlich der Verlässlichkeit der auszutauschenden Informationen herrscht. So kann sichergestellt werden, dass weder zu viele noch zu wenige Informationen bereitgestellt werden. Zusätzlich ist über den LOD eines Bauteils aber auch festgelegt, wie verlässlich die Informationen zu diesem Bauteil sind. Es ist also durchaus möglich, in einem niedrigen LOD mehr Informationen als nötig zu liefern. Durch die Angabe, dass diese Informationen in einem niedrigen LOD vorliegen, ist für den Empfänger klar, dass diese Informationen nicht als final angesehen werden dürfen und Änderungen möglich sind.

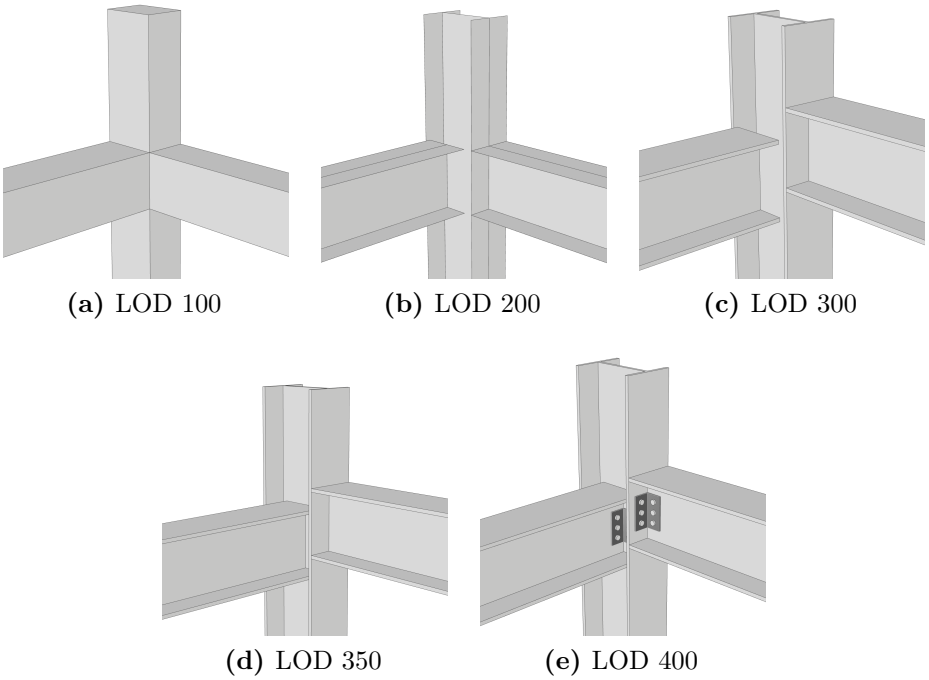
Es ist anzumerken, dass die LODs durch das BIMForum zwar grundsätzlich beschrieben werden, aber durchaus noch ein gewisser Spielraum bei der Interpretation bestehen bleibt.

Neben der fundamentalen Definition der LODs wurden daher für ausgewählte Bauteile auch konkrete Beispiele hinsichtlich der Umsetzung definiert. Ein solches Beispiel ist in Abbildung 2.7 in Anlehnung an die Definition des BIMForums dargestellt.

Im Folgenden sind die sechs LODs der allgemeinen Spezifikation des BIMForum kurz beschrieben:

- LOD 100: Das Modellelement kann im Modell mit einem Symbol oder einer anderen generischen Darstellung geometrisch beschrieben werden. Informationen, die sich auf das Modellelement beziehen, können gegebenenfalls auch nur von anderen Modellelementen abgeleitet werden. Alle Informationen müssen als Näherungswerte betrachtet werden. Die Darstellung in Abbildung 2.7 ist damit genauer als eigentlich notwendig.
- LOD 200: Das Modellelement wird innerhalb des Modells grafisch als generisches Objekt mit ungefähren Angaben zur Größe, Form, Position und Ausrichtung dargestellt. Ab LOD 200 können dem Modellelement auch nicht-grafische Informationen beigefügt werden. Alle Informationen müssen als Näherungswerte betrachtet werden.
- LOD 300: Das Modellelement wird innerhalb des Modells grafisch als spezifisches Objekt in Bezug auf Menge, Größe, Form, Position und Ausrichtung dargestellt. Das Modellelement ist in Bezug auf das Projektkoordinatensystem genau positioniert.
- LOD 350: Dieser LOD entspricht im Wesentlichen dem LOD 300. Allerdings müssen zusätzlich die Schnittstellen (Anschlüsse etc.) zu anderen Objekten im Modell enthalten sein.
- LOD 400: Das Modellelement wird innerhalb des Modells als spezifisches Objekt mit allen notwendigen Informationen für die Fertigung und Installation des Bauteils dargestellt.
- LOD 500: Das Modellelement stellt den Wie-Gebaut-Zustand nach der Bauausführung dar und wurde auf der Baustelle hinsichtlich seiner Größe, Abmessungen, Form, Position und Orientierung überprüft. Dabei ist zu beachten, dass der LOD 500 keine konkrete Definition hinsichtlich der zulässigen Toleranzen zwischen Modellelement und realem Objekt zulässt.

Die LOD-Definition des BIMForums beschreiben den Ausarbeitungsgrad eines Modellelements sowohl hinsichtlich der geometrischen Detaillierung als auch hinsichtlich der semantischen Informationen. Dies wird oft über die Formel $LOD = LOG + LOI$ verdeutlicht. Wie in der oben aufgeführten Auflistung erkennbar, sind in der allgemeinen und bauteilübergreifenden Spezifikation allerdings kaum Definitionen hinsichtlich der semantischen Informationen enthalten.



(a) LOD 100 **(b)** LOD 200 **(c)** LOD 300

(d) LOD 350 **(e)** LOD 400

Abbildung 2.7: Darstellung der Levels of Development entsprechend der Definition des BIMForums ohne den LOD 500. Dargestellt ist ein Knotenpunkt einer Rahmenkonstruktion aus Stahlstützen.

Betrachtet man die bauteilspezifischen LODs, die im umfangreichen Katalog des BIMForums enthalten sind, so ist die Semantik aber teils durchaus festgelegt. Für ein Bauteil aus Beton sind dem entsprechenden Modellelement beispielsweise ab LOD 300 Informationen hinsichtlich der Betoneigenschaften zuzuweisen. Welche Eigenschaften dies genau sind, ist aber letztlich stark projektabhängig. Das BIMForum liefert zwar im zweiten Teil seiner LOD Spezifikation auch detaillierte Vorschläge zur Definition von alphanumerischen Informationen, allerdings sind diese explizit nicht mit bestimmten LODs verknüpft.

Level of Information

Mit dem LOI (auch alphanumerischer/semantischer Detaillierungsgrad) kann definiert werden, welche Semantik ein Modell enthalten muss. Dies kann im Gegensatz zur Geometrie sehr konkret über eine Auflistung der Eigenschaften, die für einen bestimmten Bauteiltyp zu einem bestimmten Zeitpunkt vorhanden sein müssen, festgelegt werden. Der LOI kann damit als ein Bestandteil des LODs verstanden werden.

Wichtig dabei ist, dass eine Eigenschaft während der Modellierung sowohl angelegt, als auch mit einem korrekten Wert versehen werden sollte. Dies ist rein modellierungstechnisch im Vergleich zur Geometrie weniger anspruchsvoll, hängt aber letztlich immer vom verwendeten Autorenwerkzeug ab. Anders ausgedrückt besteht die Herausforderung vor allem darin, für

ein Projekt eine sinnvolle Festlegung zu treffen, welche Eigenschaften ein Modellelement zusätzlich zu seiner Geometrie haben muss und was diese Eigenschaften genau beschreiben. Zusätzlich müssen sich alle Projektbeteiligten konsequent an eine einheitliche Nomenklatur für die verwendeten Eigenschaften halten.

Die Festlegung der Eigenschaften erfolgt in der Praxis heute meist über Tabellen, in denen definiert ist, welche Eigenschaften ein Modellelement entsprechend seines Typs aufweisen muss. Zusätzlich kann festgelegt werden, wie die Werte der Eigenschaften beschaffen sein müssen, also ob es sich um einen Text oder eine Zahl handelt und ob es eine Liste vordefinierter Werte gibt, die verwendet werden müssen. Die für ein Bauteil notwendigen Attribute müssen dabei nicht zwangsläufig einem bestimmten LOI zugeordnet sein, da eine einheitliche projektübergreifende Definition nicht unbedingt zielführend ist. In der Praxis wird daher der Begriff LOI teils gar nicht konkret verwendet, sondern projektspezifisch festgelegt, welche Eigenschaften zum Ende einer Planungsphase vorliegen müssen. Daher wird auch in Frage gestellt, ob es insgesamt sinnvoll ist, hinsichtlich der Semantik überhaupt von *Levels* zu sprechen (Abualdenien *et al.*, 2021). Viel wichtiger wäre es an dieser Stelle, eine zumindest nationale Festlegung hinsichtlich der Nomenklatur der wichtigsten Eigenschaften zu treffen.

Level of Geometry

Über den Level of Geometry wird einerseits ausgedrückt, wie genau die geometrische Beschreibung eines Modellelements durch ein Symbol, ein zweidimensionales Objekt oder einen Volumenkörper in einem Modell erfolgen muss. Zusätzlich sagt der LOG dabei aber auch aus, wie verlässlich diese Informationen sind. Der LOG kann damit als ein Bestandteil des LODs verstanden werden.

Dies entspricht dem Versuch, die zunehmende Maßstabsgenauigkeit in konventionellen zeichnerischen Darstellungen über die zunehmende geometrische Detaillierung (von symbolisch über vereinfacht bis hin zu detailliert) eines Modellelements abzubilden. Die Steigerung des Detaillierungsgrads eines BIM-Modells kann damit an die fortschreitende Tiefe der Planung entlang der Leistungsphasen im Entwurfsprozess gekoppelt werden. Über die Definition eines einheitlichen LOG als Teil des LOD wird analog zum LOI die Möglichkeit geschaffen, konkrete Aussagen hinsichtlich der zu einem bestimmten Zeitpunkt geforderten geometrischen Detaillierung eines Modells zu treffen.

Die verschiedenen LOG werden in der Regel über eine textuelle Beschreibung definiert, wie sie auch in den oben aufgeführten LODs enthalten ist. Zusätzlich zu dieser allgemeinen Beschreibung liefert die LOD-Spezifikation des BIMForums für eine Vielzahl von Bauteiltypen konkrete Beispiele hinsichtlich der geometrischen Detaillierung, die ein bestimmter LOD mindestens erfordert (siehe auch Abbildung 2.7). Aufgrund der Vielzahl der Bauteile, die

in einem Bauwerk vorhanden sein können, ist diese Liste jedoch nicht als final anzusehen. Nichtsdestoweniger dienen diese Beispiele auch als Grundlage für die notwendige geometrische Detaillierung dort nicht aufgeführter Bauteile. Dies hat zur Folge, dass Planer gerade bei weniger häufig vorkommenden Bauteilen immer wieder aufs neue entscheiden müssen, wie sie diese modellieren, um den Anforderungen eines bestimmten LOD zu genügen.

Hierin liegt letztlich eine große Herausforderung bei der Erstellung eines BIM-Modells. Der Planer oder der Konstrukteur hat einerseits ein BIM-Autorenwerkzeug zur Verfügung, das es ihm erlaubt, Standardbauteile aus einer Objektbibliothek zu benutzen. Dadurch entsteht möglicherweise in einer frühen Phase des Entwerfens ein scheinbar detailliertes Modell, wenn vordefinierte geometrisch komplexe Objekte mit einem hohen LoD verwendet werden. LoD ist hier im Kontext der Verwendung in der Computergrafik zu verstehen, und soll ausdrücken, dass das modellierte Objekt eine sehr exakte Repräsentation des realen Bauteils ist. Unabhängig davon ist der LOD dieses Modells aber zu Beginn der Planung eher niedrig und erfordert keinen hohen LOG. Das Modell wirkt also deutlich genauer als es eigentlich ist, wenn es nicht im Kontext der durch den niedrigen LOD definierten Unsicherheit betrachtet wird. Während ein fachlich versierter Anwender den LOD des Modells in der Regel berücksichtigen wird, stellt ein Modell mit vielen Objekten in einem hohen LoD allerdings auch eine nicht unerhebliche Herausforderung an die Hardware des Geräts dar, auf dem das Modell erstellt oder betrachtetet wird. Je nach Datenformat in dem das Modell gespeichert wird, steigt gegebenenfalls auch die Dateigröße erheblich. Die Folge ist dann ein Modell, das ein flüssiges Arbeiten durch seine Detailtiefe beeinträchtigt.

Demgegenüber besteht die Möglichkeit für den Planer, ein sehr abstraktes Modellelement zur Modellierung eines Bauteils zu verwenden. Dadurch ist rein durch die Visualisierung des Modells zu erkennen, dass Bauteile in einem niedrigen LOD vorliegen und die Anforderungen an die Grafikhardware bleiben überschaubar. Für den Planer stellt sich nun aber die Frage, wie die Detaillierung der abstrakten Modellelemente im Zuge der weiteren Planung praktisch erfolgt. Eine Möglichkeit besteht darin, das Modellelement zu löschen und an seiner Stelle ein detaillierteres Modellelement zu erstellen. Dies kann bei umfangreichen Modellen jedoch mit einem erheblichen Aufwand verbunden sein.

Alternativ ist es denkbar, Modellelemente zu konzipieren, die von vorn herein mehrere LOGs beinhalten und dabei parametrisch so konzipiert sind, dass sich Änderungen auf einem niedrigen LOG automatisch auf die höheren LODs übertragen. Ein solches Modell wird als mehrskaliges Modell bezeichnet (Borrmann *et al.*, 2012; Jubierre *et al.*, 2013). Das Konzept der mehrskaligen Modellierung sagt aus, dass ein bestimmtes Objekt in einem Modell in verschiedenen Detaillierungsgraden vorliegt.

2.2.3 Mehrskalige Modelle

Das Prinzip der mehrskaligen Modellierung oder Mehrfachrepräsentation wird beispielsweise im Bereich der Computergrafik sowie in der Stadtmodellierung mit CityGML und der Geoinformatik allgemein eingesetzt (Kolbe, 2009; Oosterom *et al.*, 1995; Sester, 2001). In der Computergrafik wird der Begriff „mehrskalige Modellierung“ allerdings nicht verwendet, sondern nur von Level of Detail gesprochen (Luebke *et al.*, 2003). Die hierfür verwendeten LoDs wurden bereits in Abschnitt 2.2.1 eingeführt.

Sester (2001) bezeichnet die mehrskalige Modellierung im Bereich der Geoinformationssysteme (GISs) als *multiple Repräsentation* und definiert das Prinzip wie folgt: *„Multiple Repräsentation in GIS beschäftigt sich mit der Erfassung, Speicherung, Verwaltung, Analyse und Präsentation räumlicher Objekte, welche die gleichen Realweltobjekte darstellen, in einem gemeinsamen Informationssystem.“* (Sester, 2001, S. 25). Diese räumlichen Objekte sind das Resultat verschiedener Sichten auf die Realität, die durch unterschiedliche Repräsentationen der Realität entstehen. Mehrskalige Modellierung ist damit die Repräsentation von real existierenden Entitäten auf verschiedenen Abstraktionsebenen in einem Modell. Damit diese verschiedenen Repräsentationen tatsächlich ein mehrskaliges Modell bilden, muss sichergestellt werden, dass jedes Objekt den korrespondierenden Objekten auf anderen Abstraktionsstufen eindeutig zugeordnet ist.

In der 3D-Stadtmodellierung setzt CityGML dieses Konzept so um, dass es vor allem darauf zugeschnitten ist, deskriptive Modelle der bestehenden Realität zu erstellen (Kolbe *et al.*, 2021). Es werden also in der Realität bereits existente Objekte abstrahiert, um diese in verschiedenen LoDs abzubilden und für unterschiedliche Anwendungsfälle zu nutzen. Die Grundlage der Abstraktion ist in diesem Fall vergleichsweise statisch. Ein existierendes Gebäude verändert sich in der Regel nur selten in einem Maße, das eine erneute Abstraktion zur Aktualisierung des Modells erforderlich machen würde. Möchte man das Konzept der mehrskaligen Modellierung im Bauwesen umsetzen, ist zu beachten, dass BIM primär darauf zugeschnitten ist, präskriptive Modelle zu erstellen, die beschreiben, wie die Realität werden soll (Kolbe *et al.*, 2021). Der damit einhergehenden Dynamik der Modelle während der Planung eines Bauwerks wird in der Definition des Ausarbeitungsgrads eines Modells entsprechend Rechnung getragen – allerdings eben ohne dass mehrere LODs simultan vorgehalten werden. Der wesentliche Unterschied zwischen der mehrskaligen Modellierung im Kontext von Stadtmodellen und dem LOD-Konzept im Bauwesen besteht also darin, dass in einem Modell bzw. in einem Datenbestand unterschiedliche Repräsentationen für ein reales Objekt vorgehalten werden. In einem BIM-Modell ist ein Bauteil hingegen standardmäßig nur in einem LOD repräsentiert.

Ein allgemein anerkannter standardisierter Ansatz zur Speicherung und Nutzung mehrskaliger Modelle im Bereich des Building Information Modeling wurde bislang nicht eingeführt.

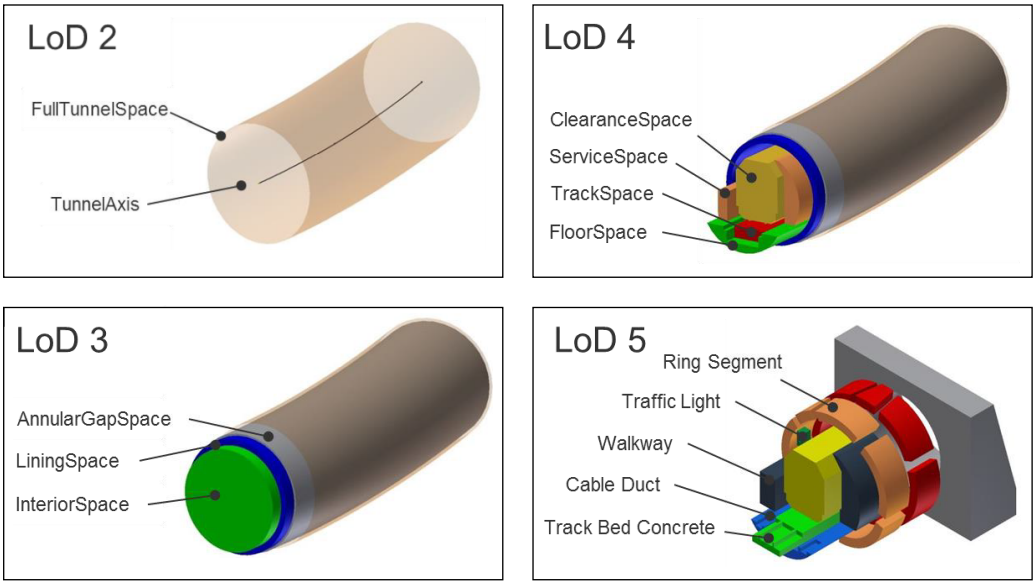


Abbildung 2.8: Das im Rahmen der DFG-Forschergruppe 3DTracks eingeführte LoD-Konzept zur Beschreibung von Schildtunneln. Während in den LoDs 2-4 Räume definiert sind, beinhaltet LoD 5 physische Objekte. *Quelle: Borrmann* et al.*, 2013a.*

Ying *et al.* (2021) schlagen jedoch beispielsweise die Nutzung eines mehrskaligen Haustechnik-BIM-Modells zur Verwendung im Bau- und Facility Management vor. Das mehrskalige BIM-Modell teilt sich hier in Mikro-, Makro- und eine schematische Ebene auf, um in umfangreichen Projekten einerseits einen Gesamtüberblick für die Bauleitung zu ermöglichen, sowie die Fachplaner bei der Koordination von Details zu unterstützen. Dabei findet allerdings keine vollkommene Integration der verschiedenen Ebenen in einem Modell statt. Es werden lediglich verschieden detaillierte Modelle innerhalb einer Anwendung vorgehalten, die ein Nutzer je nach Bedarf anzeigen kann. Die Autoren halten aber fest, dass ihr Ansatz zukünftig konkret hinsichtlich der Umsetzbarkeit mittels der Industry Foundation Classes untersucht werden solle. Hierbei ist anzumerken, dass es durch die in den IFC definierte Möglichkeit, für ein semantisches Objekt mehrere geometrische Repräsentationen vorzuhalten, grundsätzlich denkbar ist, verschieden detaillierte geometrische Repräsentationen eines Modellelements zu speichern. Bei der Konzeption des IFC-Datenmodells wurde damit dem Umstand Rechnung getragen, dass es für unterschiedliche Anwendungsfälle sinnvoll sein kann, verschiedene Geometriebeschreibungen zu verwenden (Borrmann *et al.*, 2021a).

Hervorzuheben ist an dieser Stelle ein weiterer Vorschlag zur mehrskaligen Modellierung im Bauwesen, der im Rahmen der DFG-Forschergruppe 3DTracks (Breunig *et al.*, 2017) entwickelt wurde. Hier wurde anhand von U-Bahn-Tunneln untersucht, wie ein kollaborativer Planungsprozess ablaufen kann. Ein wichtiges Forschungsthema war dabei die Mehrskaligkeit großer Infrastrukturprojekte. Da sich Infrastrukturprojekte wie Straßen, Tunnel oder Bahnstrecken über mehrere Kilometer erstrecken, müssen bei der Planung und

Erstellung der Bauwerksmodelle unterschiedliche Maßstäbe betrachtet werden: Während bei der Planung des übergeordneten Streckenverlaufs einerseits mehrere Kilometer betrachtet werden müssen, ist bei der Konstruktion von Anschlusspunkten oder Rettungsschächten eine Betrachtung im Zentimeterbereich notwendig (Borrmann *et al.*, 2015).

In Anlehnung an das in CityGML verwendete LoD-Konzept wurde daher ein semantisches Modell zur Beschreibung von Schildtunneln in fünf LoDs eingeführt (Borrmann *et al.*, 2013a; Jubierre, 2016), das es erlaubt, den typischen Ablauf einer Planung beginnend mit der Trassierung auf dem niedrigsten LoD hin zu detaillierteren Aspekten abzubilden (siehe Abbildung 2.8). Um der Dynamik eines kollaborativen Entwurfsprozesses Rechnung zu tragen, wurden zusätzlich Abhängigkeiten zwischen den verschiedenen LoDs definiert, um Änderungen in einem niedrigen LoD in die höheren LoDs zu übertragen, sodass die Konsistenz des mehrskaligen Modells sichergestellt ist. Weiterhin wurde im Rahmen des Projekts ein Vorschlag zur Erweiterung der Industry Foundation Classes zur mehrskaligen Modellierung von Tunneln ausgearbeitet (Jubierre, 2016; Vilgertshofer *et al.*, 2016b).

Verschiedene Ansätze zur Sicherung der Konsistenz mehrskaliger Stadtmodelle werden in Gröger *et al.* (2004) beschrieben. Zu beachten ist dabei, dass hier in erster Linie Szenarien untersucht werden, bei denen grobe Repräsentation durch Generalisierungsmechanismen aus Repräsentationen in höheren LoDs erzeugt werden.

Es werden dort drei wesentliche Fälle für die Art der Beziehungen zwischen den jeweiligen Repräsentationen eines Objekts in verschiedenen Detaillierungsgraden unterschieden:

- Ein bestimmtes Objekt wird in jedem LoD als eigenständige Repräsentation aufgeführt. Dies entspricht der Modellierung diskreter LoDs in der Computergrafik.
- Die Beziehung zwischen den Repräsentationen eines Objekts in den verschiedenen LoD ist hierarchisch. D.h. die Repräsentation eines Objekts in einem LoD ist in genau einer Repräsentation in dem nächst niedrigeren LoD enthalten.
- Die Beziehungen sind nur partiell hierarchisch. Es gibt also nicht zwischen allen LoD-Stufen eine hierarchische Beziehung zwischen den Repräsentationen.

Im zweiten und dritten der genannten Fälle werden dabei Beziehungen zwischen den verschiedenen Repräsentationen hergestellt, die jedoch nur für eine Konsistenz im Fall der Visualisierung genutzt werden können. Die Beziehungen ermöglichen es aber noch nicht, die Konsistenz eines mehrskaligen Modells, in dem Änderungen vorgenommen werden, zu sichern. Denn so lange die Änderung nur in einem einzelnen LoD vorgenommen wird, bleiben die Repräsentationen innerhalb aller anderen LoDs unverändert. Um die Konsistenz des Modells wiederherzustellen, müssen also alle Repräsentationen, die vom veränderten Modellelement abhängig sind und in einem anderen LoD liegen, manuell angepasst werden. In Volz (2006) wird die Problematik von Mehrfachrepräsentationen in Geoinformationssystemen weiter

untersucht, wobei das Konzept der Arbeit darin besteht *„Mehrfachrepräsentationen über explizite Relationen miteinander zu verbinden“*.

Obwohl diese Konzepte vor allem für weitgehend statische 3D-Stadtmodelle ausgelegt sind und somit nicht direkt auf dynamische Modelle, wie sie bei der Gebäudeplanung genutzt werden, angewandt werden können, bildet das Prinzip von Relationen zwischen den Repräsentationen innerhalb verschiedener LODs einen Ansatzpunkt für weitere Untersuchungen.

Vor dem Hintergrund dieser bereits existierenden Veröffentlichungen zur Konsistenzsicherung wurden in der bereits genannten DFG-Forschergruppe 3DTracks wesentliche neue Erkenntnisse erzielt. Der Kern des Ansatzes besteht im Einsatz des Konzepts der parametrischen Modellierung und entsprechender CAD-Systeme, der die Definition von Abhängigkeiten zwischen der Geometrie von Modellelementen in verschiedenen Detaillierungsstufen erlaubt. Damit ist es möglich, Modelle im Falle von Änderungen automatisiert zu aktualisieren, indem die Änderungen in die höheren LoDs propagiert werden. Die Änderung an einem Modellelement, das die Repräsentation eines Objekts in einem bestimmten LoD darstellt, wird so in allen abhängigen Repräsentationen in anderen LoDs nachgeführt, sodass das Modell in sich konsistent bleibt. Die Erstellung der entsprechenden parametrischen Modelle ist allerdings eine komplexe, arbeitsaufwändige und damit auch fehleranfällige Tätigkeit (Borrmann *et al.*, 2015).

Auch wenn das LOD-Konzept im Bauwesen immer mehr an Bedeutung gewinnt, ist hier die Integration von Methoden zur LOD- bzw. LOG-übergreifenden Konsistenzsicherung in ein Modell nur bedingt zu beobachten. Abualdenien *et al.* (2019) entwickelten beispielsweise einen Ansatz, um die Konsistenz zwischen verschiedenen Stufen der Detaillierung eines Modells während des Entwurfsprozesses zu validieren, stellen jedoch fest, dass der Detaillierungsprozess weiterhin manuell erfolgen muss. Prinzipiell kann das im 3DTracks-Projekt entwickelte Vorgehen aber durchaus auf andere Bereiche des Bauwesens übertragen werden, indem Bauteile in den verschiedenen durch das BIM Forum definierten LODs modelliert werden. Wenn hier die Geometrie der verschiedenen LODs parametrisch so miteinander verknüpft ist, dass sich Änderungen sowohl auf höhere als auch auf niedrigere LODs auswirken, können Planer je nach Anwendungsfall zwischen den verschiedenen LODs wechseln. Dies setzt aber voraus, dass bei der Erstellung eines Modells die entsprechenden parametrischen Zusammenhänge in die einzelnen Modellelemente integriert werden und das verwendete Autorenwerkzeug diese Integration einerseits unterstützt und zusätzlich entsprechende Visualisierungsmöglichkeiten für die verschiedenen LODs zur Verfügung stellt. Zur Formalisierung der Modellierungsschritte zur Erstellung solcher parametrischen Modellelemente mit einer integrierten Konsistenzsicherung bietet beispielsweise der in dieser Arbeit vorgestellte Ansatz eine gute Lösung (siehe Kapitel 7).

2.3 Ansätze zur computerorientierten Unterstützung von Entwurfs- und Konstruktionsprozessen

Entwurf bzw. Design kann als eine der komplexesten menschlichen Tätigkeiten bezeichnet werden, da meist eine Vielzahl von teils konkurrierenden Randbedingungen betrachtet werden, um eine optimale Lösung zu finden (Bhatt *et al.*, 2013). Um Herausforderungen, die sich während eines Entwurfsprozesses ergeben, zu bewältigen, können Konzepte aus dem Bereich der Entwurfsautomatisierung und Methoden des Knowledge-based Engineering (KBE) eingesetzt werden. Auch Methoden zur Automatisierung oder Unterstützung von Konstruktionsprozessen können hier Anwendung finden. In den folgenden Abschnitten wird ein Überblick über die Vorarbeiten in diesen Themengebieten vorgestellt, um die dort eingesetzten Methoden im Rahmen des in dieser Arbeit entwickelten Ansatzes einzuordnen. Dabei wird zuerst auf eine (Teil-)Automatisierung des konstruktiven Teils eines Entwurfsprozesses eingegangen und anschließend auf Verfahren, bei denen eine Unterstützung oder (Teil-)Automatisierung der Generierung von Entwürfen untersucht wird. Im Vordergrund stehen hierbei Verfahren, die sich auf Graphen und Graphtransformationen stützen.

An dieser Stelle ist anzumerken, dass ein modellbasiertes[10] Entwerfen nicht ausschließlich positiv betrachtet wird. Gerade in der Anfangsphase der Verwendung von BIM und der dadurch notwendigen BIM-Modelle, wurde eine mögliche Einschränkung der Freiheit des architektonischen Entwurfs als kritisch angesehen. Dies lässt sich rückblickend auch durch die anfänglich noch vergleichsweise unausgereiften BIM-Autorenwerkzeuge erklären. Diese Thematik wurde beispielsweise von Kiviniemi *et al.* (2009) ausführlich untersucht. Auch Tovey (1989) stellte schon sehr früh im Kontext der einsetzenden Verbreitung von CAD-Systemen fest, dass diese Systeme inhärent ungeeignet zur Erstellung innovativer Entwürfe sind, da sich der kreative Prozess immer innerhalb der begrenzten Möglichkeiten des genutzten Systems bewegen muss. Diese Aussage deckt sich auch mit den Ergebnissen eines von van Amstel *et al.* (2016) durchgeführten Experiments, bei dem Studierende mit einem parametrischen Entwurfswerkzeug arbeiten. Hier wird allerdings auch festgestellt, dass die Breite möglicher Entwürfe letztlich von vielen verschiedenen Randbedingungen abhängen, die sich nicht unbedingt negativ auf das Ergebnis auswirken. Denn wie Tovey (1989) feststellt, birgt die Nutzung eines (parametrischen) CAD-Systems die Möglichkeit Entwürfe zu optimieren, indem rechnerbasiert schnell eine Vielzahl möglicher Varianten erzeugt wird.

Die im Folgenden vorgestellten Themengebiete beschränken sich allerdings nicht auf einen Entwurfsprozess, bei dem ein geometrisches Modell im Vordergrund steht. Die verschiedenen Ansätze haben vielmehr trotz der teils sehr unterschiedlichen Methoden das übergeordnete Ziel, ein möglichst optimales Ergebnis als Resultat eines Entwurfsprozesses zu erreichen.

[10] Damit ist ein Entwurfsprozess, bei dem ein digitales Bauwerksmodell verwendet wird, gemeint.

Neben konkreten geometrischen Modellen kann dieses Ergebnis auch von sehr abstrakter semantischer Natur sein und einen Entwurf nicht anhand seiner Geometrie, sondern beispielsweise durch verschiedene Kennzahlen oder eine bestimmte Struktur charakterisieren.

2.3.1 Automatisierung von Modellierungs- und Entwurfsprozessen

Es steht außer Frage, dass ein zielführender Entwurfsprozess ohne die Verwendung von Modellen generell kaum denkbar und gerade im Bauwesen nicht praktisch umsetzbar wäre. Wie bereits in Abschnitt 2.1.3 beschrieben, wurden und werden dafür insbesondere im Hinblick auf die verfügbaren (parametrischen) CAD-Werkzeuge stetig neue Lösungen durch die Softwareindustrie entwickelt. Und auch wenn bei aktuellen CAD-Werkzeugen nach wie vor eine manuelle Modellierung erforderlich ist, wird diese im Vergleich zu älteren Versionen durch viele Funktionen unterstützt, bei denen es sich bereits um eine Automatisierung handelt. Diese werden durch den Nutzer allerdings nicht unbedingt als solche wahrgenommen, da sie bereits zur Gewohnheit geworden sind. Als Beispiele seien hier das Kopieren von Elementen oder die Rückgängig-Funktion genannt.

Im Folgenden sollen aber die Möglichkeiten in Bezug auf vergleichsweise komplexe oder komplizierte Konstruktionsprozesse im Kontext der (parametrischen) Modellierung betrachtet werden. Mit komplexen und komplizierten Konstruktionsprozessen sind hier nach Schuh (2014) Prozesse gemeint, bei denen die Dynamik der Änderungen und/oder die Anzahl der betrachteten Elemente hoch ist (Obergrießer, 2016). Hierbei kann entweder nur die Automatisierung der Konstruktion eines Modells oder auch das automatische Generieren von (zufälligen) Entwurfsvarianten, also einer tatsächlichen Entwurfsautomatisierung, betrachtet werden. Eine scharfe Abgrenzung ist allerdings nicht in allen Fällen möglich. Dies bedingt sich vor allem dadurch, dass in der Literatur teils schon eine automatisierte parameter- und wissensbasierte Erstellung von geometrischen Modellen als Entwurfsautomatisierung bezeichnet wird.

Hinsichtlich der Automatisierung von Konstruktionsprozessen unterscheidet Obergrießer (2016) nach Spur *et al.* (1993) und Abulawi (2012) zwischen zwei grundlegenden Ansätzen, dem vollautomatisierten und dem teilautomatisierten Konstruktionsprozess. Der wesentliche Unterschied besteht darin, dass bei teilautomatisierten Konstruktionsprozessen eine interaktive Steuerung durch den Nutzer erfolgt, während bei einem vollautomatisierten Prozess nach anfänglicher Eingabe von Informationen (Eingangsparameter, Randbedingungen etc.) keine Notwendigkeit zur manuellen Einflussnahme besteht. Dies schließt allerdings die Möglichkeiten, welche sich durch eine interaktive Steuerung ergeben, aus. Dadurch verringert sich die Flexibilität des Prozesses, da der Algorithmus, der der Automatisierung zugrunde liegt, meist nicht alle Eventualitäten abdecken kann. Insofern ist ein vollautomatisierter

[10] Die Begriffe Konstruktions- und Modellierungsprozess sind hier als synonym anzusehen.

Konstruktionsprozess nur für Prozesse mit überschaubaren Randbedingungen, also beispielsweise für Teilprozesse eines teilautomatisierten Konstruktionsprozesses wirtschaftlich umsetzbar (Obergrießer, 2016).

Ein wesentlicher Vorteil teilautomatisierter Konstruktionsprozesse liegt insofern in ihrer Flexibilität, da bestimmte richtungweisende oder kritische Entscheidungen weiterhin manuell durch den Nutzer getroffen werden können (Abulawi, 2012). Aktuelle CAD-Werkzeuge bieten daher meist standardmäßig integrierte Funktionen an, mit denen repetitive Teile eines Konstruktionsprozesses als parameterabhängige Funktionen definiert und anschließend abgerufen werden können (z. B. iParts in Autodesk Inventor oder Musterelemente in Siemens NX). Diese Teilautomatisierung unterliegt aber natürlich immer den durch die Benutzeroberfläche des CAD-Werkzeugs definierten Randbedingungen. Eine erweiterte Möglichkeit besteht in der Nutzung von Makros (Yang *et al.*, 2013), mit deren Hilfe eine starre Abfolge von Konstruktionsschritten programmiert werden kann (Obergrießer, 2016). Durch die Nutzung der Programmierschnittstelle (API) einer CAD-Software kann eine weitere Flexibilisierung erfolgen, die verschiedene Randbedingungen oder Nutzereingaben während der Laufzeit berücksichtigt. Hierbei wird vollkommen unabhängig von der Benutzeroberfläche auf die Klassen und Methoden eines Softwareprodukts zugegriffen, was entsprechende Programmierkenntnisse voraussetzt. Trotz der grundsätzlich vielfältigen Möglichkeiten, die sich durch die beschriebenen Ansätze realisieren lassen, ist allerdings anzumerken, dass diese zwangsläufig auf ein CAD-Werkzeug eines bestimmten Herstellers ausgelegt sind und nicht ohne Weiteres in andere Programme übertragen werden können.

Bevor im Folgenden auf die Entwurfsautomatisierung eingegangen wird, ist aber noch zu erwähnen, dass automatisierte und teilautomatisierte Prozesse zur Erstellung von Modellen gerade im Hinblick auf die Nutzung Digitaler Zwillinge (siehe Seite 19) während des Betriebes von Gebäuden und Bauwerken zunehmend an Bedeutung gewinnen. Aktuell beschäftigt sich eine Vielzahl von Forschungsprojekten mit Ansätzen, die den Erstellungsprozess der geometrisch-semantischen Bauwerksmodelle als Teil eines Digitaler Zwilling automatisieren sollen (Dang *et al.*, 2021; Lu *et al.*, 2020b; Lu *et al.*, 2020c; Lu *et al.*, 2019; Mafipour *et al.*, 2021; Pan *et al.*, 2021; Stojanovic *et al.*, 2019). Zusätzlich wird die Rekonstruktion von Gebäuden zur Erstellung von Stadtmodellen auch in der Geoinformatik untersucht (Steinhage, 2001).

In jedem Fall ist sowohl bei teil- als auch bei vollautomatisierten Prozessen unumgänglich, dass das für die Automatisierung notwendige (ingenieurtechnische) Wissen explizit formuliert werden kann (Johannsen *et al.*, 1991). Mit explizitem Wissen ist eine Form von Wissen gemeint, die einfach formalisiert und dadurch weitergegeben werden kann (Frappaolo, 2008). Dies ist eine der wesentlichen Voraussetzungen für die erfolgreiche Umsetzung von

[10] Application Programming Interface

Knowledge-based Engineering[11]. Auch wenn der Begriff des KBE bisher nicht vollumfänglich und abschließend definiert ist, schließt er sowohl die Automatisierung von Kontruktions- als auch von Entwurfsprozessen mit ein. Dies wird auch in den Übersichtsartikeln von Cooper *et al.* (2007), Reddy *et al.* (2015) und Verhagen *et al.* (2012) zu diesem Thema festgehalten.

Der Begriff Entwurfsautomatisierung beschreibt im Kontext dieser Arbeit einen Prozess des Entwerfens, der durch Programme und Software-Tools unterstützt wird und über die reine Automatisierung von Konstruktionsprozessen hinausgeht. Im Normalfall findet auch dabei keine vollkommene Automatisierung statt, da grundlegende Entscheidungen nach wie vor manuell durch einen verantwortlichen Ingenieur getroffen und vor allem überprüft werden müssen (Amadori, 2012). Vielmehr werden in erster Linie repetitive oder triviale Aspekte eines Entwurfsprozesses unterstützt, um so beispielsweise mehr Freiraum für einen kreativen bzw. ingenieurtechnischen architektonischen Entwurf zu schaffen. Insofern kann die (Teil-)Automatisierung von Konstruktionsprozessen als Voraussetzung für die Entwurfsautomatisierung betrachtet werden. Weiterhin kann durch die computergestützte Erstellung von Entwürfen eine größere Bandbreite an möglichen Lösungen erzeugt und anschließend bewertet werden (Shea *et al.*, 2005). Hierbei taucht in vielen Arbeiten der Begriff der *computational design synthesis* auf, mit dem letztlich aber auch das computergestütze und damit (teil-)automatisierte (Neu-)Entwickeln von Entwurfsvarianten beschrieben wird (Cagan *et al.*, 2005; Helms, 2013). Im Folgenden wird ein Überblick über die in den letzten Jahrzehnten entwickelten Ansätze im Bereich der Ingenieurwissenschaften gegeben. Im nachfolgenden Abschnitt werden entsprechende Verfahren betrachtet, bei denen Graphen als Datenstruktur zur Anwendung kommen.

Beispielsweise entwickelten Havemann *et al.* (2004) eine Methode, mit der das Maßwerk (das geometrische Muster) gotischer Fenster auf Basis einer *Generative Modeling Language* prozedural erstellt werden kann. Die Autoren nutzen grundlegende geometrische Muster, die parametrisiert und durch einen modularisierten Ansatz kombiniert werden. Wesentliche Themen sind die Identifikation der grundlegenden Modellierungsoperationen und die Untersuchung, wie parametrisierte Konstruktionsabläufe präzise in einer formalen Sprache ausgedrückt werden können.

Im Bereich des KBE entwickelten Amadori *et al.* einen Ansatz zur *geometriebasierten Entwurfsautomatisierung*[12], bei dem durch automatisierte Erstellung und Veränderung die Geometrie der Entwürfe von Luftfahrzeugen untersucht werden (Amadori, 2012; Amadori *et al.*, 2012). Dabei wurden sogenannte *High Level CAD templates* in CATIA[13] V5 als

[11] Knowledge-based Engineering (KBE) ist ein Forschungsfeld, das sich mit Methoden und Technologien zur Erfassung und Wiederverwendung von Wissen aus der Produkt- und Prozessentwicklung beschäftigt (Verhagen *et al.*, 2012).

[12] *Geometry Based Design Automation*

[13] CATIA ist ein CAD-System der französischen Software-Entwicklungsunternehmens Dassault Systèmes.

parametrische Modelle erstellt, die mittels der Logik einer *inference engine* instantiiert werden und als Basis für iterative Entwurfsvorgänge dienen.

Zur automatisierten Erzeugung von Geometrien werden oft auch sogenannte *Shape Grammars*[14] eingesetzt, die in ihren Produktionsregeln Ingenieurwissen abbilden können. Hoisl *et al.* entwickelten in diesem Bereich eine Grammatik, die die visuelle und interaktive Definition und Ausführung von Produktionsregeln zulässt (Hoisl *et al.*, 2011a, 2011b; Hoisl, 2012). Diese Grammatik basiert auf parametrisierten Primitiven, sowie parametrischen und nichtparametrischen Regeln, mit denen diese Primitive angeordnet werden können. Der wesentliche Beitrag liegt in der Möglichkeit, die in anderen Ansätzen meist statischen Regeln interaktiv und visuell erstellen und anpassen zu können. Granadeiro *et al.* (2013) nutzen Shape Grammars als Basis ihres Ansatzes zur Optimierung eines generativen Entwurfssystems, während Ruiz-Montiel *et al.* (2013) Shape Grammars in Kombination mit *reinforcement learning* verwenden, um eine große Anzahl von Entwurfsvarianten innerhalb bestimmter Randbedingungen zu erzeugen. Auch Mata *et al.* (2019) kombinieren Shape Grammars mit Methoden der parametrischen Modellierung zur Erzeugung von Entwurfsvarianten, wobei der Fokus hier auf der Konzeption von Regeln liegt, in die ästhetische Randbedingungen und eine möglichst positive Wahrnehmung durch Konsumenten eingebettet wird.

Auch im Bereich des Bauwesens wurden Ansätze zur automatisierten Erstellung von Entwürfen erforscht. Beispielsweise entwickelte Singer (2014) einen Prototypen für den Einsatz von Knowledge-based Engineering in frühen Phasen des Brückenentwurfs und Donaubauer *et al.* (2016) stellten prozedurale Verfahren zur automatischen Generierung von 3D-Infrastrukturobjekten (Brücken und Tunnel) auf Basis von digitalen Geländemodellen vor. Auch in den Ansätzen von Hartung *et al.* (2020) und Girardet *et al.* (2021) werden Modelle von Brückenbauwerken automatisiert auf Basis parametrischer Modelle erzeugt, wobei Hartung *et al.* (2020) den Fokus auf die Modellierung von Bestandsbauwerken legt. Im Gegenzug zielt der Ansatz von Girardet *et al.* (2021) auf die Erstellung eines parametrischen Basismodells ab, mit dem eine Vielzahl von Brückenentwürfen generiert werden kann. Im Bereich der Architektur von Gebäuden schlägt Caneparo (2022) eine Methode vor, um das Wissen über Objekte, räumliche Beziehungen und Randbedingungen von einem generativen Prozess zu Erstellung von Grundrissen zu trennen. Erwähnenswert sind hier auch die Arbeiten von Boonstra *et al.* (Boonstra, 2020; Boonstra *et al.*, 2020, 2021), die sich mit dem räumlichen Entwurf von Gebäuden in frühen Entwurfsphasen mittels sog. *Design Grammars* auseinandersetzen.

Neben Ansätzen zur wissenbasierten Erstellung von Instanzmodellen gibt es auch Untersuchungen zur automatisierten Erstellung von Produktdatenmodellen[15]. Hier nutzen Lee *et al.* (2006a) einen linguistischen Ansatz zur Entwicklung einer formalen Methode, mit der

[14] Shape Grammars sind Produktionssysteme, die zur Generierung von geometrischen Figuren genutzt werden (Stiny, 1975).

[15] Datenmodell (M2) entsprechend der OMG-Metamodell-Hierarchie.

Wissen und spezifische Informationen in den Bereichen Design, Konstruktion, Fertigung und Instandhaltung gesammelt und zur (teil-)automatisierten Konzeption von Datenmodellen genutzt werden kann.

Erwähnenswert sind auch Ansätze, die eine automatisierte Bewertung von Entwürfen ermöglichen. Die Problemstellung liegt hier in der Vielzahl von Richtlinien, Randbedingungen und Regeln, die bei Entwurfsprozessen beachtet werden müssen. Ziel ist es, solche Regeln automatisiert zu überprüfen (Eastman *et al.*, 2009) oder auch zu interpretieren (Niemeijer *et al.*, 2014). Die Kombination einer solchen automatisierten Regelprüfung mit einer automatisierten Erstellung einer Vielzahl von Entwurfsvarianten birgt letztlich die Möglichkeit, diese Varianten direkt zu analysieren und hinsichtlich ihrer Regelkonformität zu bewerten.

Der in dieser Arbeit vorgestellte Ansatz ordnet sich insofern in die vorgestellten Entwicklungen ein, als dass auch hier Ingenieurwissen in Form von Konstruktionsregeln eines parametrischen Modellierungsprozesses formalisiert wird, um es später abzurufen und in einem Entwurfsprozess einzusetzen. Da in dieser Arbeit Graphen und Graphersetzungssysteme eine wesentliche Rolle spielen, werden im folgenden Abschnitt entsprechende Ansätze dezidiert betrachtet.

2.3.2 Graphbasierte Ansätze zur Automatisierung von Modellierungs- und Entwurfsprozessen

Die Nutzung von Graphen, Graphersetzungssystemen und Graphgrammatiken zur Erstellung von Entwürfen werden bereits von Maher (1990) und Mullins *et al.* (1990) beschrieben. Diese Techniken werden seitdem in einer Vielzahl von Ansätzen im Bereich der Ingenieurwissenschaften eingesetzt (Kolbeck *et al.*, 2021). Unabhängig davon ob nur Graphersetzungssysteme zu Anwendung kommen oder Graphgrammtiken verwendet werden, um eine Vielzahl von möglichen Entwürfen auf Basis eines Graphersetzungssystems zu generieren, liegt allen Ansätzen zugrunde, dass ein Graph zur Repräsentation des im Rahmen des Entwurfs erstellten Modells verwendet wird.

Beispielsweise nutzen Schaefer *et al.* (2005) eine Graphgrammtik zum Entwurf von Satelliten. Hier wird der Zustand eines Entwurfs durch einen *design graph* repräsentiert, der mittels der entwickelten Grammatik erzeugt und anschließend ausgewertet werden kann, um ein Modell zu analysieren und zu visualisieren. Die Produktionsregeln der Grammatik erlauben es dabei, vordefinierte Modifizierungsoperationen anzuwenden und den Entwurf dadurch zu verändern oder weiterzuentwickeln. Ein ähnlicher Ansatz wird in Zhao *et al.* (2020) verfolgt, wobei hier der Entwurf von Robotern für bestimmte Geländearten betrachtet wird.

Auch zur Unterstützung des kreativen Prozesses in frühen Entwurfsphasen existieren bereits graphbasierte Konzepte, bei denen beispielsweise die Generierung und Visualisierung von

zweidimensionalen Grundrissen und dreidimensionaler Gebäudestrukturen im Vordergrund stehen (Grabska *et al.*, 2012; Ślusarczyk *et al.*, 2021). Im Ansatz von Langenhan *et al.* (2013) wird eine erste Skizze eines Grundrisses als Graph repräsentiert, der anschließend verwendet wird um ähnliche Grundrisse in bereits bestehenden Gebäuden zu identifizieren und so den Entwurfsprozess zu unterstützen. Für den konzeptionellen Gebäudeentwurf wurden weiterhin an der RWTH dazu im Rahmen des ConDes-Projekts Software-Werkzeuge konzipiert, die mit Hilfe eines graphbasierten Ansatzes einerseits konzeptionelle Skizzen und andererseits konzeptionelles Wissen abbilden (Kraft *et al.*, 2007a, 2007b; Kraft *et al.*, 2006). Gan (2022) nutzt einen Graph als Datenmodell für die Repräsentation wesentlicher Merkmale modularer Gebäude. Ziel des Ansatzes ist es, den generativen Entwurf für modulare Bauweisen zu automatisieren und zu optimieren, in dem die graphbasierte Repräsentation verändert und die erzeugten Alternativen anschließend hinsichtlich ihrer Tauglichkeit bewertet werden.

In einem Artikel von Belhaouari *et al.* (2014) wird ein weiterer interessanter Ansatz für die regelbasierte geometrische Modellierung vorgestellt. Mit *Jerboa* wurde ein topologiebasiertes Modellierungstool entwickelt, das geometrische Objekte durch eine graphenbasierte Datenstruktur definiert, die jedes topologische Element mit seiner geometrischen Form verknüpft. Die Modellierungsoperationen werden hier nicht als Funktionen einer Software programmiert, sondern in Form von speziellen Graphersetzungsregeln implementiert, die in einem graphischen Editor verändert und damit für verschiedene Modellierungszenarien angepasst werden können. Es werden allerdings nur die tesselierten Oberflächen von Körpern durch Flächen, Kanten und Punkte als Graph beschrieben. Graphersetzungsregeln werden dann beispielsweise benutzt, um diese Oberflächen zu glätten. Parametrische oder prozedurale Modellierungsoperationen werden nicht betrachtet.

Kniemeyer (2008) nutzt eine Graphgrammatik, um automatisiert eine Vielzahl von realistischen Pflanzenmodellen zu erzeugen. Auch wenn in dieser Arbeit nicht direkt Ingenieurwissen in einer Graphgrammatik formalisiert wird, gibt diese Arbeit einen ausführlichen Einblick in die Entwicklung eines Graph-Metamodells zur Repräsentation der Modelle und in das detaillierte Vorgehen zur Konzeption von Graphersetzungsregeln, mit denen Erweiterungen und Verfeinerungen eines Modells beschrieben werden. Weiterhin werden in dieser Arbeit auch die theoretischen Grundlagen der Graphersetzung ausführlich beschrieben.

Weiterhin erwähnenswert sind die Ansätze von Grabska, Ślusarczyk *et al.* (Grabska *et al.*, 2018; Ślusarczyk *et al.*, 2017a; Ślusarczyk *et al.*, 2017b). Hier wird anhand von Beispielen aus dem Grundriss- und Brückenentwurf untersucht, wie Graphstrukturen genutzt werden können um Wissen formal zu repräsentieren. In diesen Arbeiten wird einerseits untersucht, wie Entwürfe graphbasiert erstellt und verändert werden können. Weiterhin wird betrachtet, wie sich die Randbedingungen einer ingenieurtechnischen Fragestellung generell als Graph repräsentieren lassen, um diese Graphen anschließend als Datenstruktur zur Lösungsfindung zu nutzen.

Auch im Bereich der bereits erwähnten *computational design synthesis* werden in einer Vielzahl von Ansätzen Graphersetzungssysteme und Graphgrammatiken eingesetzt. Chakrabarti *et al.* (2011) stellen die Entwicklungen bis 2011 vor, und beschreiben in weiten Teilen auch graphbasierte Ansätze. In den Arbeiten von Helms *et al.* ist ein graphbasierter Ansatz zum konzeptionellen Entwurf von Produktmodellen im Maschinenbau ausgearbeitet (Helms *et al.*, 2009; Helms *et al.*, 2012). Insbesondere in seiner Dissertation geht Helms (2013) auf die Verwendung von Graphgramatiken zur Erzeugung von verschiedenen Varianten eines Entwurfs ein. In einer der vorgestellten Grammatiken werden beispielsweise die verschiedenen Komponenten des Antriebsstrangs eines Hybridantriebs als Elemente eines Graphen formalisiert und mittels der Graphgrammatik zu physikalisch funktionsfähigen Entwürfen zusammengesetzt. Auch in den Arbeiten von Münzer *et al.* werden Graphen im Kontext der *computational design synthesis* verwendet Muenzer *et al.* (2017), Münzer (2016) und Münzer *et al.* (2013). Hier werden allerdings die generierten Entwürfe automatisiert bewertet und durch Randbedingungen auf valide Lösungen beschränkt. Alle Ansätze in diesem Bereich haben allerdings gemein, dass sie vergleichsweise abstrakte Entwurfsvorgänge betrachten und insofern keine konkreten geometrischen Modelle als Ergebnisse erzeugen.

Auch ein Artikel von Eichhoff *et al.* (2016) ist in den Bereich der *computational design synthesis* einzuordnen, wobei der Fokus nicht auf einem konkreten Anwendungsfall liegt, sondern vielmehr übergeordnet darauf abzielt, automatisierte Entwurfssysteme zu erstellen. Hier werden zwei Ansätze vorgestellt, die zum Ziel haben, graphbasierte Produktionsregeln automatisiert aus beispielhaften Graphen abzuleiten und diese zur Erstellung konzeptioneller Entwürfe zu verwenden. Auch die Ansätze von Königseder *et al.* (Königseder, 2015; Königseder *et al.*, 2015) zielen darauf ab, die Erstellung generativer Systeme durch eine Unterstützung der Regelentwicklung zu vereinfachen, wobei auch graphbasierte Systeme betrachtet werden.

Zusammenfassend kann festgestellt werden, dass eine Vielzahl von graphbasierten Ansätzen zur Lösung ingenieurtechnischer Fragestellungen im Kontext der Unterstützung und Automatisierung von Modellierungs- und Entwurfsprozessen erarbeitet wurden. Die Ansätze reichen dabei von sehr konkreten graphbasierten Repräsentationen geometrischer Objekte, die regelbasiert verändert werden, bis hin zum Einsatz von Graphen, die eine Fragestellung sehr abstrakt anhand ihrer konzeptionellen Bestandteile betrachten. Der in dieser Arbeit vorgestellte Ansatz ist als vergleichsweise konkret einzuordnen, da die Geometrie von parametrischen Modellen so als Graph repräsentiert wird, dass sie direkt aus dem Graph abgeleitet werden kann. Die Regeln, mit denen die Veränderungen des Modells beschrieben werden, können dabei sowohl konkret im Sinne einzelner kleinteiliger Modellierungsoperationen definiert werden, als auch Sequenzen von Modellierungsschritten beschreiben, mit denen konzeptionelle Entwurfsschritte formalisiert werden können. Weiterhin ist anzumerken, dass keine Literatur zu Ansätzen bekannt ist, mit denen parametrische

Modelle und die Modellierungsschritte zur Erstellung dieser Modelle als Graph repräsentiert werden.

2.4 Zusammenfassung und Einordnung im Kontext dieser Arbeit

Im vorangegangenen Kapitel wurden die wesentlichen Hintergründe, die der Motivation für diese Arbeit zugrunde liegen, vorgestellt. Dies betrifft übergeordnet die zunehmende Etablierung von BIM und die damit einhergehende Einführung des modellbasierten Arbeitens im Bauwesen. Gerade dieses Arbeiten mit semantisch-geometrischen digitalen 3D-Bauwerksmodellen birgt ein großes Potenzial zur Verbesserung der Planungs- und Ausführungsprozesse im Bauwesen. Dies geht jedoch mit verschiedenen Herausforderungen hinsichtlich der Anforderungen an die Softwareprodukte, der Kompetenzen der Beteiligten und der notwendigen Änderungen etablierter Prozesse einher.

Im Kontext dieser Arbeit sind hier vor allem die beschriebenen notwendigen Entwicklungen hinsichtlich einer herstellerunabhängigen graphbasierten Formalisierung von Modellierungswissen hervorzuheben. Dieses Modellierungswissen kann zur Detaillierung von Modellen generell, sowie insbesondere zur gleichzeitigen Sicherstellung der Konsistenz solcher Modelle, die Bauteile in unterschiedlichen Detaillierungsgraden enthalten, genutzt werden. Hierzu werden Methoden der parametrische Modellierung verwendet. Der in dieser Arbeit beschriebene Ansatz zur Unterstützung eines modellbasierten Entwurfs- und Modellierungsprozesses greift bestehende Entwicklungen in diesem Bereich auf, um letztlich eine Unterstützung dieser Prozesse durch die Formalisierung von Modellierungswissen zu ermöglichen.

Eine solche Formalisierung des Modellierungswissens innerhalb eines parametrischen Entwurfsprozesses wurde bisher noch nicht hinsichtlich einer herstellerneutralen und strukturierten Umsetzung betrachtet. Zu untersuchen, ob und wie hierfür eine graphbasierte Repräsentation genutzt werden kann, die unabhängig von bestimmten proprietären Softwaresystemen und Dateiformaten ist, stellt somit die wesentliche Forschungsfrage dar, die in dieser Arbeit behandelt wird.

Kapitel 3

Geometrische und parametrische Modellierung

Wird ein Gebäude- oder Bauwerksmodell als BIM-Model bezeichnet, lässt sich heute daraus in der Regel ableiten, dass es sich um ein digitales dreidimensionales Modell handelt, das sowohl Geometrie als auch Semantik beinhaltet, wobei die semantische Repräsentation führend ist (Borrmann *et al.*, 2021a). Während die Semantik des Modells durchaus Informationen und Zusammenhänge, die ohne eine geometrische Repräsentation auskommen, beschreiben kann, sind geometrische Objekte, die keiner semantischen Entität zugeordnet werden können, nicht sinnvoll nutzbar. Nichtsdestoweniger ist die Verwendung von dreidimensionaler Geometrie eine der wesentlichen Voraussetzungen für die erfolgreiche Umsetzung von Building Information Modeling (Borrmann *et al.*, 2021b).

Der geometrische Teil eines BIM-Modells besteht aus der Beschreibung der geometrischen Repräsentation der semantischen Entitäten und ermöglicht damit eine Visualisierung eines Modells. Diese geometrischen Repräsentationen sind in der Regel Volumenkörper, durch die das Erscheinungsbild des jeweiligen Modellelements beschrieben wird. Neben diesen dreidimensionalen geometrischen Repräsentationen werden insbesondere bei Modellelementen mit einem niedrigen geometrischen Detaillierungsgrad (LOG) auch zwei- oder eindimensionale Repräsentationen verwendet. Den wesentlichen Anteil eines BIM-Modells machen aber Volumenkörper aus, die je nach dem LOG der Modellelemente, die sie repräsentieren, einer mehr oder weniger starken Abstraktion des geplanten bzw. gebauten realen Bauteils entsprechen (siehe auch Abschnitt 2.2.2). Mit Hilfe der geometrischen Modellierung können die einzelnen

Volumenkörper erstellt und zu einem Modell zusammengefügt werden. Weiterhin können den geometrischen Objekten semantische Eigenschaften als Attribute hinzugefügt werden.

Zur Erstellung von BIM-Modellen werden computergestützte Methoden und digitale Werkzeuge eingesetzt. Die computergestützte Modellierung dreidimensionaler geometrischer Objekte bildet daher eine wesentliche Grundlage für die Erstellung von BIM-Modellen. Dabei kommen auch Methoden der parametrischen Modellierung zum Einsatz, die eine höhere Flexibilität bei der Modellierung ermöglichen, dafür aber auch tiefergehende Kenntnisse der entsprechenden Methoden bei den Anwendern voraussetzen.

Gegenstand dieser Arbeit ist die Entwicklung eines Graphersetzungssystems zur Unterstützung eines parametrischen Modellierungsprozesses. Daher sind die geometrische Modellierung, die Beschreibung der entsprechenden Modelle und insbesondere die Methoden der parametrischen Modellierung für diese Arbeit relevant. In diesem Kapitel werden daher die wesentlichen Begriffe und Definitionen des Themenfelds der computergestützten geometrischen und parametrischen Modellierung eingeführt.

Dazu wird einerseits allgemein auf die computergestützte Erstellung und Repräsentation geometrischer Modelle eingegangen. Anschließend wird das Paradigma der parametrischen Modellierung detailliert vorgestellt und hinsichtlich des in dieser Arbeit entwickelten Ansatzes eingegrenzt. Daraus kann letztlich abgeleitet werden, wie die Erstellung von Modellen entsprechend dieses Paradigmas durch ein Graphersetzungssystem unterstützt werden kann.

3.1 Geometrische Modellierung von Volumenmodellen

Auch wenn zum Zeitpunkt der Drucklegung dieser Arbeit nach wie vor in vielen Bauprojekten zweidimensionale Konstruktionsmethoden zur Planungen von Gebäuden und Bauwerken eingesetzt werden, ist mit der voranschreitenden Umsetzung von BIM ein Paradigmenwechsel hin zu dreidimensionalen Planungsmethoden unverkennbar. Dabei ist die konsequente Nutzung eines Building Information Models, das die für die Planung, den Bau und den Betrieb eines Gebäudes relevanten Informationen beinhaltet, unabdingbar. Die dreidimensionale Gebäudegeometrie ist ein wesentlicher Bestandteil eines BIM-Modells, ohne den viele der BIM-Anwendungen nicht umsetzbar wären. Zur Beschreibung dieser dreidimensionalen Geometrie werden vornehmlich Volumenmodelle, die sich aus Volumenkörper zusammensetzen, eingesetzt. Diese Volumenmodelle können mit verschiedenen Verfahren erstellt und gespeichert werden.

Ein solches Volumenmodell (engl. *solid*) beschreibt die Oberfläche und das Volumen eines realen physikalischen Körpers. Hierbei kann zwischen direkten und indirekten Darstellungsschemata unterschieden werden (Bungartz *et al.*, 2002; Vajna *et al.*, 2018). Direkte Darstellungsschemata beschreiben das Volumen selbst, während bei den indirekten Darstel-

lungsschemata die Beschreibung über Kanten und Oberflächen erfolgt. Ein solche Oberfläche muss ich aus geschlossenen Teilflächen zusammensetzen, die eine Orientierung besitzen, sodass zwischen Innen und Außen unterschieden werden kann.

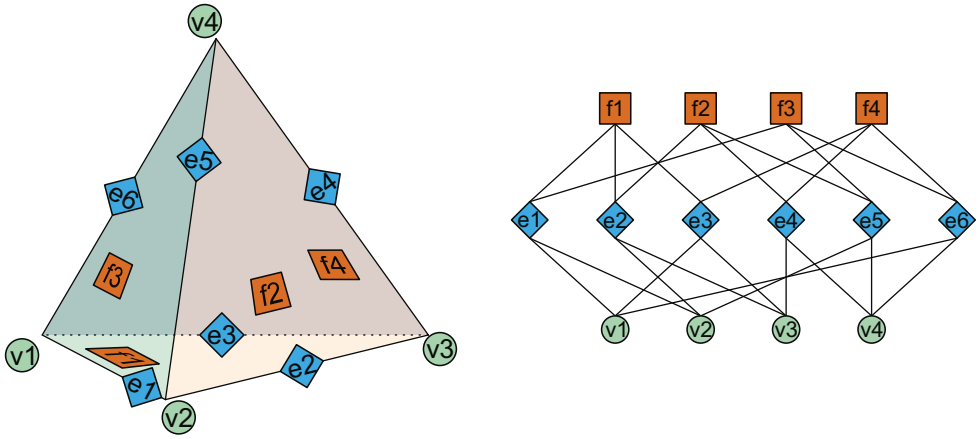
Möchte man hingegen nicht nur die Darstellung und sondern auch den Prozess der Modellierung von Volumenkörpern betrachten, können die verschiedenen Verfahren in akkumulative und generative Modelle (Romberg, 2005) bzw. explizite und implizite Ansätze aufgeteilt werden (Borrmann *et al.*, 2021b; Obergrießer, 2016). Wird ein Volumenkörper explizit beschrieben, so wird die Oberfläche bzw. das Volumen eines Körpers als finales Resultat eines Modellierungsprozesses beschrieben. Der Körper wird dann beispielsweise indirekt über seine Oberfläche beschrieben. Diese Oberfläche setzt sich aus Teilfächen zusammen, deren Eckpunkte durch Koordinaten definiert sind. Bei einer Beschreibung mittels eines impliziten Verfahrens wird hingegen die Folge der Konstruktionsschritte aufgezeichnet, sodass der Ablauf des Modellierungsprozesses nachvollziehbar bleibt. Hier wird auch oft von einer prozeduralen oder historien-basierten Darstellung gesprochen. Beachtenswert ist hierbei, dass eine implizite Darstellung vergleichsweise einfach in eine explizite Darstellung überführt werden kann, während dies umgekehrt nur mit erheblichem Aufwand und nicht zwangsläufig eindeutig möglich ist.

3.1.1 Explizite Verfahren

Die einfachste Form der expliziten Beschreibung eines Volumenmodells besteht darin, die Oberfläche des Körpers zu *tesselieren*. Die Oberfläche wird dabei in der Regel durch ein Netz aus Dreiecken beschrieben (trianguliert, siehe Abbildung 3.3b). Während Körper mit vollkommen planaren Oberflächen mit dieser Methode exakt beschrieben werden können, ist bei gekrümmten Oberflächen lediglich eine Annäherung möglich. Mit zunehmender Größe der Dreiecke nimmt die Genauigkeit dieser Annäherung jedoch ab. Werden allerdings sehr viele und sehr kleine Dreiecke verwendet, ist die Annäherung für viele Anwendungsfälle ausreichend genau. Mit zunehmender Anzahl der Dreiecke erhöht sich allerdings auch der Speicheraufwand. Die Beschreibung eines Körpers durch ein Dreiecksnetz wird beispielsweise für reine Visualisierungsanwendungen oder für numerische Berechnungen und Simulationen verwendet (Borrmann *et al.*, 2021b).

Eine verbesserte Darstellung ist durch die Nutzung der *Boundary Representation Methode* (B-rep oder BRep) möglich (siehe Abbildung 3.3a), bei der es sich um eine der gängigsten Methoden zur rechnerbasierten Beschreibung von dreidimensionalen Körpern handelt (Borrmann *et al.*, 2021b). Auch hier werden die Körper über ihre Hüllfläche beschrieben, die sich aus benachbarten Flächen (engl. *faces*) zusammensetzt (Stroud, 2006). Diese Flächen werden durch Kanten (engl. *edges*) begrenzt, die über ihre Anfangs- und Endknoten (engl. *vertex* bzw. *vertices*) definiert sind. Die Hierarchie aus Flächen, Kanten und Knoten, die

die Oberfläche eines Körpers beschreiben, stellen die Topologie des modellierten Körpers dar, während die Koordinaten der Knoten und der Verlauf der Kanten die Geometrie beschreiben. Dabei beschreibt die Topologie die Struktur eines Körpers während die Geometrie die Form beschreibt (Stroud, 2006). Die Topologie kann beispielsweise mit einem *Vertex-Edge-Face*-Graphen beschrieben werden. Durch diesen vef-Graph wird festgehalten, durch welche Anfangs- und Endknoten sich eine Kante definiert, welche Kanten eine Fläche begrenzen und welche Flächen die Oberfläche eines Körpers bilden.



solid	faces
1	1,2,3,4

face	edges
1	1, 2, 3
2	2, 4, 5
3	1, 5, 6
4	3, 4, 6

vertex	coordinates
1	2, 0, 0
2	0, 0, 0
3	3, 0, 0
4	1, 1, 3

edge	vertices
1	1, 2
2	2, 3
3	3, 1
4	3, 4
5	2, 4
6	1, 4

Abbildung 3.1: Eine einfache BRep-Datenstruktur gefüllt mit Daten zur Beschreibung einer Pyramide. Der Vertex-Edge-Face-Graph beschreibt die Beziehungen zwischen Knoten, Kanten und Flächen und damit die Topologie des Körpers. Die Koordinaten der Knoten legen die Geometrie fest. *Quelle: Borrmann* et al.*, 2021b.*

Bei der Beschreibung der Geometrie kann nach Stroud (2006) zwischen numerischer und analytischer Geometrie unterschieden werden. Bei der Verwendung analytischer Geometrie ist die Form konkret festgelegt. Werden hierbei nur gerade Kanten zugelassen, ist es ausreichend, die Koordinaten der Knoten zu speichern, durch die sich die Geometrie des Körpers definiert. Auch in diesem Fall können wie bei der Verwendung von Dreiecksnetzen nur planare Flächen beschrieben werden. Zusätzlich können aber auch kreisförmige Kanten analytisch definiert werden. Um auch die Abbildung komplexerer gekrümmter Oberflächen zu ermöglichen, ist es notwendig, numerische Geometrien zu verwenden. Hier werden Freiform-Kurven (Bézier-Kurven, B-Splines, NURBS) zur Beschreibung der Kanten und analoge Methoden zur Flächenbeschreibung für die Repräsentation gekrümmter Flächen eingesetzt. Der Verlauf wird dabei nicht explizit sondern implizit durch sog. Kontrollpunkte definiert.

Dazu sind neben den Koordinaten dieser Kontrollpunkte auch zusätzliche Informationen zum Verlauf der Kurven bzw. Flächen zu betrachten und zu speichern.

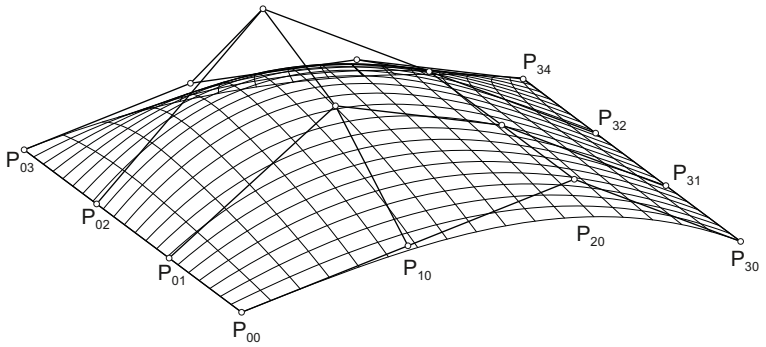


Abbildung 3.2: NURBS-Patch mit einem Feld von 5 x 4 Kontrollpunkten zur Beschreibung einer gekrümmten Oberfläche. *Quelle: Borrmann* et al.*, 2021b.*

Um auch komplexere Körper, die Aussparungen oder Hohlräume enthalten, abzubilden, muss das Datenmodell erweitert werden. Dies wird beispielsweise durch den ACIS-Modellierkern (Corp, 2021) umgesetzt. Dieser Modellierkern kommt in verschiedenen CAD- und BIM-Softwareanwendungen zum Einsatz und erlaubt es, die Form beinahe jedes denkbaren Körpers zu beschreiben (Borrmann *et al.*, 2021b).

Ein weiteres Verfahren zur expliziten Beschreibung von Volumenmodellen ist das Normzellen-Aufzählungsschema. Hier wird der zu beschreibende Raum durch ein dreidimensionales Gitter aus gleichgroßen würfelförmigen Zellen eingeteilt (siehe Abbildung 3.3c). Der gesamte Raum kann hier über eine Matrix beschrieben werden, deren Einträge festlegen, ob sich eine Zelle innerhalb oder außerhalb des zu beschreibenden Volumens befindet. Dadurch kann einerseits für einen Punkt im Raum sehr schnell abgefragt werden, ob sich dieser Punkt innerhalb oder außerhalb des Volumens befindet, andererseits hängt die Genauigkeit der Darstellung immer von der Größe der Zellen ab, sodass schräge und runde Flächen lediglich approximiert werden und sich in der Visualisierung ein Treppeneffekt zeigt (Romberg, 2005).

3.1.2 Implizite Verfahren

Um ein Volumenmodell implizit zu beschreiben, kann beispielsweise die *Constructive Solid Geometry* (CSG) eingesetzt werden. Bei diesem Verfahren werden verschiedene einfache Grundkörper (Primitive) mittels mengentheoretischen Boolescher Operationen kombiniert. Durch diese Kombinationen kann eine große Vielfalt komplexer Körper erzeugt werden.

Als vordefinierte Primitive stehen beispielsweise Quader, Zylinder, Kugeln, Torus oder Pyramiden zur Verfügung, deren Abmessungen parametrisiert sind und daher verändert werden können. Zur Kombination dieser Primitive stehen die drei Operationen Vereinigung ($\cup$), (Durch-)Schnitt ($\cap$) und Differenz ($/$) zur Verfügung. Der Aufbau eines komplexen Körpers

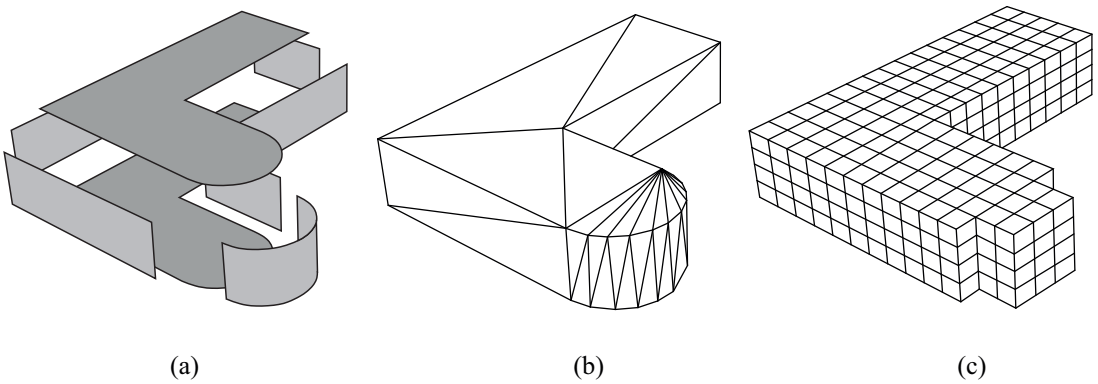


Abbildung 3.3: Ein Volumenmodell, das mittels B-rep (a), Dreicksnetz (b) und Normzellen (c) repräsentiert wird.

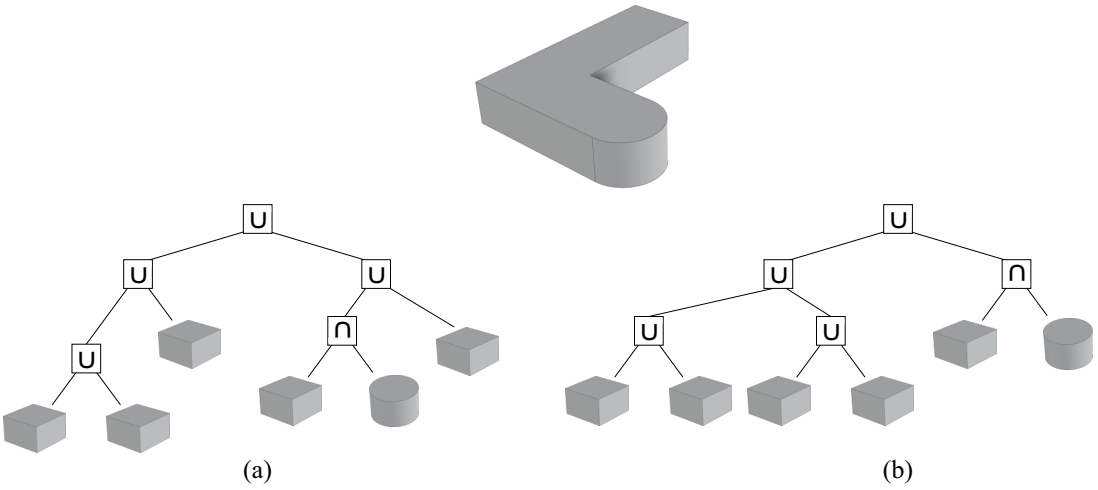


Abbildung 3.4: Der oben dargestellte Volumenkörper kann durch die in (a) und (b) dargestellten CSG-Bäume repräsentiert werden. Als Primitive werden Quader und ein Zylinder genutzt.

durch die schrittweise Kombination der Primitive mittels der verschiedenen Operationen wird in einem sog. CSG-Baum dargestellt (Abbildung 3.4), der ähnlich wie ein binärer Baum aufgebaut ist (Romberg, 2005). Die äußeren Knoten des Baums (Blätter) repräsentieren dabei die Primitive und die inneren Knoten die Operationen. Ein wesentlicher Vorteil dieses Vorgehens besteht darin, dass jeder Körper, der durch einen CSG-Baum definiert wird, auch einem tatsächlich realisierbaren physikalischen Körper entspricht. Durch den CSG-Baum wird weiterhin der Herstellungs- oder Modellierungsprozess des Körpers konkret dargestellt und ist damit nachvollziehbar. Die Topologie und Geometrie der Oberfläche des Körpers ist allerdings nicht explizit abgespeichert, sondern muss erst durch die Interpretation des CSG-Baums konstruiert und berechnet werden. Dadurch ist es beispielsweise nicht möglich Attribute an eine bestimmte Oberfläche anzuheften (Romberg, 2005). Weitere Nachteile

bestehen nach Neuberg (2004) darin, dass keine direkte grafische Ausgabe möglich ist und ein Körper durch verschiedene CSG-Bäume dargestellt werden kann, was den Vergleich von Körpern erschwert (siehe Abbildung 3.4).

Ein weiteres implizites Verfahren zur impliziten Modellierung von Volumenmodellen besteht in der Erzeugung der 3D-Geometrie durch Transformationen und Interpolationen, die teils auch gesammelt als *Sweepingverfahren* oder *Sweeping Operationen* bezeichnet werden (Obergrießer, 2016). Ein dreidimensionaler Körper wird dabei auf Basis eines zweidimensionalen Profils erzeugt. Dieses Profil wird entweder entlang einer Leitlinie oder -kurve verschoben oder um eine Achse rotiert, sodass ein 3D-Modell entsteht. Je nachdem ob das Verfahren auf Basis eines offenen oder eines geschlossenen Profils erfolgt, entsteht entweder ein Flächenmodell oder ein Volumenmodell. Es kann zwischen drei verschiedenen Verfahren unterschieden werden: *Translation*, *Rotation* und *Interpolation* (Obergrießer, 2016; Vajna *et al.*, 2018). Bungartz *et al.* (2002) verwenden hier die Begriffe Verschiebegeometrieschema und Interpolationsschema, die inzwischen aber nicht mehr gebräuchlich sind. Obergrießer (2016) verwendet für so erzeugte Modelle den Oberbegriff Sweep-Modell.

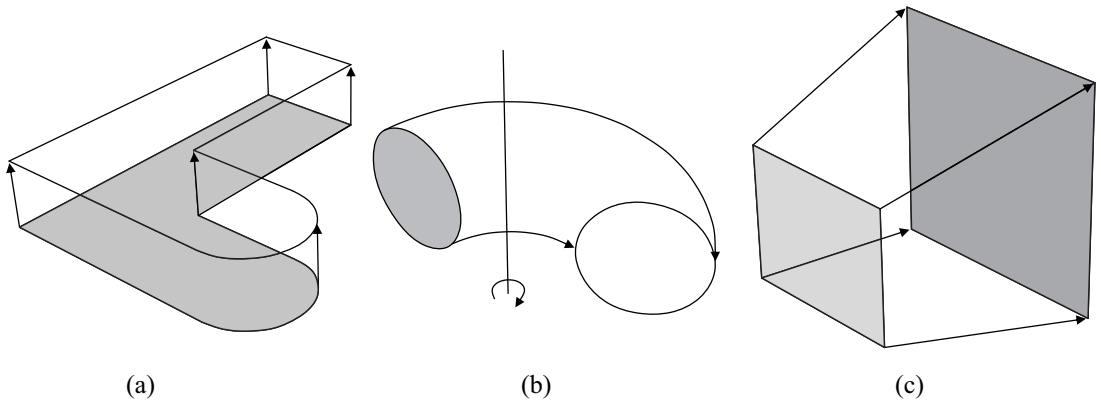


Abbildung 3.5: Volumenmodelle, die mittels Extrusion (a), einer Rotation (b) bzw. Interpolation (c) erstellt wurden.

Bei einer *Translation* wird ein in der Ebene gezeichnetes 2D-Profil entlang eines Pfades (der nicht in dieser Ebene liegt) um eine bestimmte Distanz verschoben. Der Raum, der bei dieser Verschiebung durch das Profil überstrichen wird, beschreibt ein Volumenmodell. Hierbei wird der Begriff Extrusion für eine Parallelverschiebung des Profils entlang eines linearen Pfads verwendet. Ist der Verschiebungspfad eine Kurve, wird in der Regel von einem Sweep gesprochen. Gerade bei der praktischen Umsetzung dieser Verfahren in CAD-Programmen werden die Begriffe allerdings nicht vollkommen einheitlich verwendet. Teils wird der Begriff Extrusion auch nur für eine orthogonale Parallelverschiebung[1] verwendet.

[1]Damit ist gemeint, dass der Pfad der Extrusion orthogonal zur Ebene des Profils sein muss.

Weiterhin kann ein Volumenmodell durch *Rotation* eines Profils um eine gerade oder gekrümmte Drehachse erzeugt werden. Der Pfad entlang dem das Profil geführt wird entspricht in diesem Fall einer Kreislinie. Auch hier wird der Körper durch den Raum, den das Profil im Zuge der Drehung durchläuft, aufgespannt. Über die Angabe eines Winkels kann festgelegt werden, wie weit die Rotation des Profils erfolgen soll. Um beispielsweise eine Torus zu erzeugen muss ein kreisförmiges Profil um 360° rotiert werden.

Die Erstellung eines Volumenmodells mittels des Interpolationsverfahrens erfolgt ähnlich wie bei der Translation. Der wesentliche Unterschied besteht darin, dass mehr als ein Profil definiert werden kann, das während der Translation durchlaufen wird. Dadurch verändert sich der Querschnitt des erzeugten Körpers kontinuierlich, da er sich an die durch die verschiedenen Profile definierten Randbedingungen anpasst. Es ist dabei auch möglich sehr unterschiedliche Profile wie z. B. ein Polygon und einen Kreis zu verwenden. Die Interpolation, die die Geometrie des Körpers zwischen den Profilen als fließender Übergang erzeugt, wird auch als Morphing bezeichnet (Obergrießer, 2016). Dieses Verfahren wird auch insbesondere in CAD-Anwendungen oft als *Lofting* bezeichnet.

3.1.3 Einordnung

Betrachtet man die verschiedenen impliziten und expliziten Verfahren zur Beschreibung und Modellierung von Volumenmodellen, ist klar erkennbar, dass bei den impliziten Verfahren der Prozess, mit dem das Modell erstellt wurde, deutlich besser nachvollziehbar ist. Dies zeigt sich auch dadurch, dass die einzelnen Schritte (z. B. Boolesche Operationen, Extrusionen etc.), die bei einer impliziten Repräsentation letztlich das Volumenmodell beschreiben, in vielen CAD-Anwendungen als Werkzeuge wiederfinden und sich bei der 3D-Modellierung nutzen lassen. Ein explizites Modell manuell zu erstellen, indem man beispielsweise die einzelnen Punkte des Dreiecksnetzes definiert und durch Linien verbindet, ist mit einem wesentlich größeren Aufwand verbunden und deutlich weniger intuitiv.

Durch die Nachvollziehbarkeit der Modellierungsschritte ist es bei implizit beschriebenen Modellen deutlich einfacher möglich, im Nachhinein Anpassungen vorzunehmen. Dies ist insbesondere im Hinblick auf den Austausch von Modellen ein wesentlicher Vorteil, wobei der Empfänger eines Modells hier in der Lage sein muss, die implizit beschriebenen Modellierungsschritte korrekt und eindeutig zu interpretieren. Sowohl der Sender als auch der Empfänger eines Modells müssen Softwarewerkzeuge benutzen, die geeignete Export- und Importschnittstellen zur Verfügung stellen. Da diese Schnittstellen eine hohe Komplexität aufweisen, ist die Implementierung mit einem erheblichen Aufwand verbunden (Borrmann *et al.*, 2021b). Dies geht bei explizit beschriebenen Modellen mit einem geringeren Aufwand einher, da lediglich die finale Topologie und Geometrie eingelesen werden muss und es nicht notwendig ist, die einzelnen Modellierungsschritte korrekt nachzuvollziehen und das finale

Modell so zu rekonstruieren. Nachträgliche Anpassungen sind zwar auch bei explizit beschriebenen Modellen grundsätzlich möglich, allerdings muss hierbei die Geometrie direkt editiert werden. Dies ist aufwändiger als die Änderung der konkreten Modellierungsschritte eines implizit beschriebenen Modells, da gegebenenfalls eine große Menge von Kontrollpunkten editiert werden muss. Zusätzlich muss sichergestellt werden, dass durch die entsprechenden Veränderungen keine Fehler in der Beschreibung der Oberfläche des Modells (z. B. Lücken oder Überlappungen zwischen den Teilflächen) auftreten.

Auch im Kontext dieser Arbeit liegt ein wesentlicher Vorteil der impliziten Beschreibung von Modellen in der Nachvollziehbarkeit der Modellierungsschritte, da die ursprüngliche Entwurfsabsicht des Modellautors vergleichsweise klar erkennbar ist. Über die Modellierungshistorie, die in der impliziten Beschreibung enthalten ist, kann der Ablauf nachvollzogen werden und das darin enthaltene Wissen des ursprünglichen Konstrukteurs auf andere Anwendungsfälle übertragen werden. Hierfür ist allerdings ein tiefes Verständnis der verwendeten Modellierungsmethoden notwendig, insbesondere dann, wenn die im Folgenden beschriebenen Methoden der parametrischen Modellierung eingesetzt werden.

Dies bedingt sich auch dadurch, dass die Verfahren der implizierten Modellierung als Vorläufer der parametrischen Modellierung angesehen werden (Obergrießer, 2016; Vajna *et al.*, 2018) können. Kombiniert man die Erstellung von Volumenmodellen durch Translation, Rotation etc. mit der Constructive Solid Geometry, ist es bereits möglich, sehr komplexe Volumenmodelle zu erzeugen. Die hier beschriebenen Methoden der impliziten Repräsentation von Volumenmodellen bilden damit eine wesentliche Grundlage für die parametrische Modellierung, die im nächsten Abschnitt detaillierter beschrieben wird.

3.2 Parametrische Modellierung

Die im vorangegangenen Abschnitt 3.1 aufgeführten Methoden zur Beschreibung und Erstellung digitaler geometrischer Modelle sind im Wesentlichen darauf ausgerichtet, eine statische Repräsentation eines Objektes zu beschreiben. Es wird also nur eine konkrete Instanz eines geometrischen Modells gespeichert und bei einer expliziten Repräsentation nur das Resultat der Modellierungsoperationen. Auch wenn bei einer impliziten Repräsentation die Schritte der Modellierung gespeichert werden, ist diese Form der Repräsentation nicht direkt darauf ausgelegt, im Nachhinein weitreichende Änderungen vorzunehmen. Sollen Änderungen am Modell durchgeführt werden, ist es daher in der Regel notwendig, den Modellierungsprozess zu rekonstruieren und Teile des Modells zu löschen und entsprechend der gewünschten Änderung neu zu erstellen. Dies ist in der Regel ein arbeitsintensives und zeitaufwändiges Vorgehen, das in einem iterativen Planungsprozess zu hohen Kosten führen kann. In der Retrospektive ist es daher nur verständlich, dass das Verfahren der

parametrischen Modellierung entwickelt wurde und sich in verschiedenen Ausprägungen in Softwarewerkzeugen zur Modellerstellung durchsetzt.

Verglichen mit den bereits beschriebenen Methoden der geometrischen Modellierung ist das wesentliche Merkmal des Konzepts der parametrischen Modellierung (auch parametrisch-assoziative bzw. prozedural-parametrische Modellierung), dass ein erstelltes Modell schnell verändert werden kann – ohne dass eine manuelle Neuerstellung notwendig ist und einzelne Modellierungssoperationen erneut ausgeführt werden müssen. Dabei wird die Anordnung der geometrischen Elemente über Parameter und parametrische Zwangsbedingungen so gesteuert, dass die grundlegende Beschaffenheit des Modells bestehen bleibt. Dieses Konzept wurde bereits in den 1990er Jahren im Zuge der Digitalisierung des technischen Zeichnens entwickelt (Shah *et al.*, 1995) und hat sich mittlerweile fest etabliert (Sacks *et al.*, 2004).

Die parametrische Modellierung wird in vielen kommerziellen und Open-Source-CAD-Anwendungen wie beispielsweise Autodesk Inventor, Autodesk AutoCAD, Siemens NX, Dassault Catia und FreeCAD verwendet. Erst durch die Implementierung der Methoden der parametrischen Modellierung in diesen CAD-Werkzeugen kommt die Mächtigkeit dieses Ansatzes zum Tragen: Auch wenn eine handgezeichnete Skizze mit Informationen zu topologischen Zusammenhängen der gezeichneten Elemente und Angaben zu Parametern versehen werden kann, ist es in diesem Fall offensichtlich nicht möglich, Änderungen einzelner Parameterwerte ohne manuelle Änderungen auf die vollständige Zeichnung zu übertragen. Erst wenn die in der Zeichnung enthaltenen Informationen in eine digitale Skizze übertragen werden, ist es möglich, Änderungen automatisiert zu propagieren und so von dem zusätzlichen Aufwand, der für die Erstellung eines parametrischen Modells in Kauf genommen werden muss, zu profitieren. Wird von einer parametrischen Skizze[2] oder einem parametrischen Modell gesprochen, ist insofern davon auszugehen, dass die Erstellung mittels eines parametrischen CAD-Werkzeugs erfolgt ist. Die theoretischen Konzepte der parametrischen Modellierung sind also letztlich eng mit der softwaretechnischen Umsetzung verknüpft, sodass die praktische Anwendung immer im Kontext der Implementierung betrachtet werden muss. Anders ausgedrückt ist ein parametrisches Modell immer stark von den Funktionen und Möglichkeiten der CAD-Anwendung, in der es erstellt wurde, geprägt.

Anfänglich wurde das Konzept der parametrischen Modellierung vor allem im Maschinenbau eingesetzt. Da Produkte hier meist in großen Serien gefertigt werden, müssen entsprechende Produktmodelle in einem iterativen Prozess optimiert werden, der viele Änderungen erfordert. Ist ein parametrisches Modell vorhanden, können solche Änderungen schneller und mit einem geringeren Aufwand umgesetzt werden. Obwohl auch im Bauwesen während der Planung eine Vielzahl von Änderungen an einem Entwurf vorgenommen werden, hat sich eine Nutzung von parametrischen CAD-Werkzeugen hier erst später etabliert. Aktuelle

[2] Im Kontext der parametrischen Modellierung bezeichnet der Begriff Skizze eine parametrisierte Zeichnung, das heißt eine Zeichnung deren Abmessungen auf Basis von Parametern verändert werden kann. Siehe auch Abschnitt 3.2.1.

BIM-Autorenwerkzeuge implementieren die entsprechenden Methoden aber inzwischen umfangreich, sodass sie zur Erstellung leicht anpassbarer Modelle von Gebäuden verwendet werden können. Während der Fokus anfänglich auf der Erstellung von Modellen für Hochbauprojekte lag, konnte durch verschiedene wissenschaftliche Arbeiten gezeigt werden, dass die parametrische Modellierung auch im Bereich der Infrastrukturplanung zielführend eingesetzt werden kann (Ji *et al.*, 2013; Obergrießer, 2016), sodass einzelne BIM-Autorenwerkzeuge inzwischen Erweiterungen für z. B. die Planung von Brücken bieten.

Die wesentlichen Grundlagen und Begrifflichkeiten der parametrischen Modellierung werden in den folgenden Abschnitten beschrieben. Dabei werden in erster Linie die Begriffe verwendet, die sich in gängigen CAD-Anwendungen und damit in der praktischen Anwendung etabliert haben. Eine umfassende Einführung in dieses Themengebiet findet sich im Wesentlichen in (Shah *et al.*, 1995), wobei auch jüngere Veröffentlichungen (Obergrießer, 2016; Vajna *et al.*, 2018) eine ausführliche Beschreibung des Konzepts und der Weiterentwicklung beinhalten. In Kapitel 5 werden die für diese Arbeit relevanten Methoden der parametrischen Modellierung auf Basis der im Folgenden beschriebenen Grundlagen detailliert definiert.

3.2.1 Grundlagen

Prinzipiell können sowohl zweidimensionale Zeichnungen als auch dreidimensionale Modelle mithilfe von parametrischen CAD-Werkzeugen erstellt werden. In aktuell auf dem Markt verfügbaren Werkzeugen wird hier meist ein Ansatz verwendet, mit dem auf Basis parametrischer 2D-Skizzen durch prozedurale Modellierungsoperationen Volumenmodelle erstellt werden können (Jubierre, 2016). Hinsichtlich dem Vorgehen zur Erstellung der Skizzen und Modelle wird in der Literatur zwischen dem prozedural-parametrischen und dem variationalen Modellierungsansatz unterschieden (Hoffmann *et al.*, 2002). Der wesentliche Unterschied besteht darin, dass bei Verwendung des prozedural-parametrischen Modellierungsansatzes eine genau definierte Reihenfolge der Konstruktionsschritte definiert werden muss (Berling *et al.*, 1993), während der variationale Ansatz in dieser Hinsicht flexibler ist. Obergrießer (2016) diskutiert ausführlich die Vor- und Nachteile, stellt aber letztlich fest, dass in typischen parametrischen CAD-Systemen inzwischen meist ein hybrider Ansatz verwendet wird, um die Vorteile beider Ansätze nutzen zu können. Auch das im Folgenden beschriebene Vorgehen zur Erstellung eines parametrischen Modells orientiert sich an diesem hybriden Ansatz und damit am gängigen Ablauf der Modellierung in einem parametrischen CAD-System.

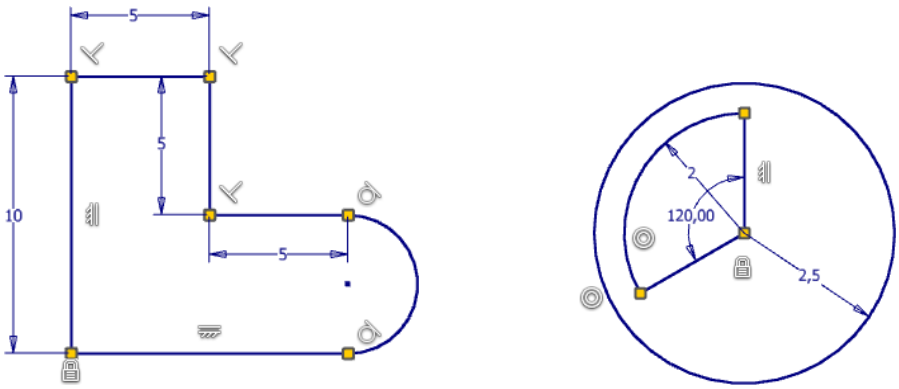
Parametrische 2D-Modelle entstehen durch die Kombination von geometrischen Elementen und Zwangsbedingungen (engl. *constraints*). Ein solches 2D-Modell wird im Kontext der parametrischen Modellierung als *Skizze* bezeichnet. Eine Skizze unterscheidet sich von einer gewöhnlichen 2D-Zeichnung dadurch, dass die Anordnung der geometrischen Elemente nicht

statisch ist, sondern angepasst werden kann. In einem parametrischen CAD-System ist eine Skizze also eine Zeichnung, bei deren Erstellung noch keine präzisen Informationen zu den tatsächlichen Dimensionen angegeben werden müssen. Stattdessen kann in einem ersten Schritt nur eine grobe Anordnung von geometrischen Elementen definiert werden. Die Anordnung dieser Elemente kann allerdings durch die Verwendung verschiedener Zwangsbedingungen genau festgelegt werden.

Im Zuge der Erstellung einer Skizze in einer parametrischen CAD-Anwendung wird ein System von geometrischen Elementen und geometrisch-topologischen Zwangsbedingungen aufgebaut. Diese Zwangsbedingungen definieren die Topologie der geometrischen Elemente und können in eine Skizze integriert werden. In Abbildung 3.6 sind zwei solche Skizzen abgebildet, wobei die Zwangsbedingungen zur Definition der Topologie (sog. geometrische Zwangsbedingungen) als graue Symbole und gelbe Quadrate dargestellt sind. Hier wird beispielsweise definiert, dass zwei Linien orthogonal sind oder die Endpunkte dieser Linien an derselben Stelle liegen. Die verschiedenen Zwangsbedingungen werden in Abschnitt 3.2.3 grundlegend beschrieben und hinsichtlich ihrer Verwendung in dieser Arbeit in Kapitel 5 ausführlich definiert.

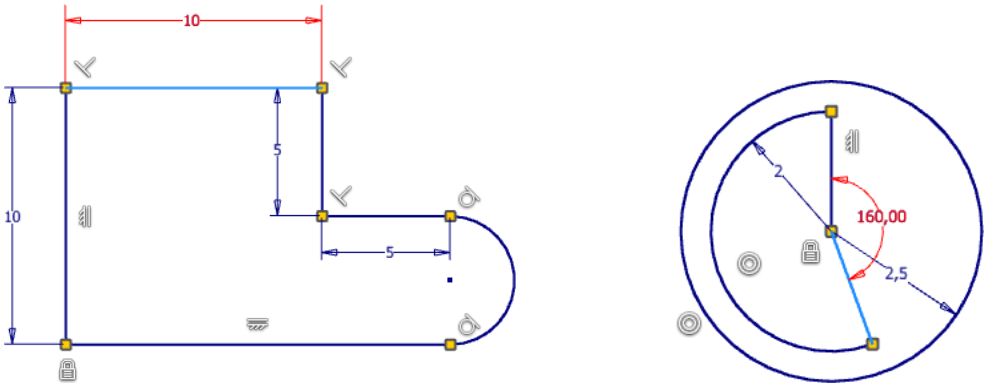
Mit Hilfe der Zwangsbedingungen können auch bestimmte Abmessungen (Abstände, Radien, Winkel etc.) durch die Verwendung von Variablen (den Parametern) gesteuert werden und sind nicht durch finale numerische Werte statisch festgelegt. Man spricht hier von dimensionalen Zwangsbedingungen, die in Abbildung 3.6 als Bemaßungslinien dargestellt sind. Über die Änderung der Werte der Parameter dieser dimensionalen Zwangsbedingungen ergibt sich die Möglichkeit, einen Entwurf schnell zu ändern. Dies ist in Abbildung 3.7 dargestellt: Hier wurden die Parameterwerte der jeweils in rot gezeichneten dimensionalen Zwangsbedingungen manuell verändert, wobei die übrige Skizze durch das CAD-Werkzeug automatisch so angepasst wurde, dass alle anderen Zwangsbedingungen weiterhin eingehalten werden. Prinzipiell können so durch die Verwendung einer Vielzahl von möglichen Werten für die einzelnen Parameter verschiedene Varianten eines Entwurfs schnell und weitgehend automatisch erzeugt und untersucht werden.

In der Regel wird zur Erstellung eines parametrischen Modells zuerst eine Skizze erstellt. Durch die Platzierung geometrischer Elemente in der Skizze wird das grundsätzliche Aussehen eines Entwurfsobjekts, z. B. der Querschnitt eines Körpers, festgelegt. Anschließend werden die nötigen Verknüpfungen zwischen den Elementen der Skizze durch geometrische Zwangsbedingungen definiert und mithilfe der dimensionalen Zwangsbedingungen flexibel gestaltet. Dabei wird durch das Modellierungssystem eine Lösung bestimmt, bei der die Anforderungen aller Zwangsbedingungen erfüllt werden (Shah *et al.*, 1995). Der Teil eines CAD-Systems, der für diesen Prozess verantwortlich ist, wird als Geometric Constraint Solver (GCS) bezeichnet und in Abschnitt 3.2.4 näher beschrieben.



(a) (b)

Abbildung 3.6: Geometrische und dimensionale Zwangsbedingungen in einer 2D-Skizze. Die geometrischen Zwangsbedingungen werden über verschieden Symbole dargestellt, die dimensionalen Zwangsbedingungen als Bemaßungen.

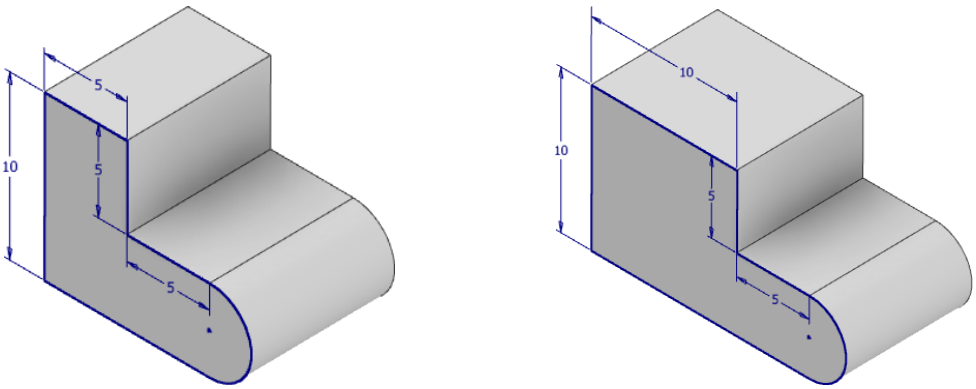


(a) (b)

Abbildung 3.7: Durch die Veränderung der Parameterwerte der rot dargestellten dimensionalen Zwangsbedingungen kann die parametrische Skizze angepasst werden. Die Topologie und damit die generelle Form der Skizze bleiben aufgrund der weiteren Zwangsbedingungen erhalten.

Eine Skizze bildet nun die Grundlage für weitere Modellierungsoperationen[3], die durch Extrusionen, Sweeps, Lofts oder boolesche Operationen 3D-Geometrie erzeugen (Borrmann *et al.*, 2012; Mun *et al.*, 2003). Auch hier können Parameter genutzt werden, damit z. B. die Länge einer Extrusion im Nachhinein anpassbar ist. So können letztendlich auch verschiedene Varianten des erzeugten 3D-Modells erstellen, indem er die Werte der Parameter ändert und so eine Neuberechnung des Modells einleitet. Durch den Modellierkern des Modellierungssystems wird dabei die Geometrie des Modells automatisch angepasst. In Abbildung 3.8 sind Extrusionen der Skizzen aus Abbildung 3.6a bzw. 3.7a dargestellt. Hier ist zu erkennen, dass sich die Veränderung eines Parameterwerts innerhalb der Skizze auch auf die danach erstellte 3D-Geometrie auswirkt. Man spricht hier von einer assoziativen Modellkopplung (Obergrießer, 2016).

[3]Diese Modellierungsoperationen werden auch als prozedurale Operationen bezeichnet.



(a) (b)

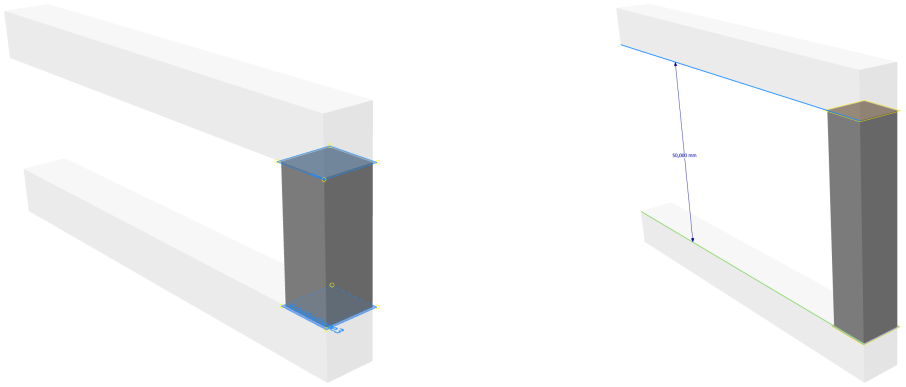
Abbildung 3.8: 3D-Geometrie, die auf Basis einer Skizze durch Extrusion erstellt wurde. Wird ein Parameterwert in der Skizze auf der linken Seite verändert, so passt sich das Modell entsprechend an (rechts).

Die auf diese Weise erstellten Volumenkörper können anschließend genutzt werden, um sie zu komplexeren Modellen zusammenzusetzen. In Abbildung 3.9 sind beispielsweise drei aus einem Quadrat extrudierte Quader dargestellt, die so angeordnet wurden, dass sie einen Rahmen bilden. Zur Anordnung der parametrisch erstellten Volumenkörper werden topologische Zwangsbedingungen (auch Assembly Constraints) eingesetzt, die in gängigen CAD-Systemen auch als Baugruppenabhängigkeiten bezeichnet werden. Eine solche topologische Zwangsbedingung kann beispielsweise als sog. *Mating-Zwangsbedingung* zwei Flächen unterschiedlicher Bauteile als komplanar definieren, sodass diese Bauteile immer aneinander anschließen. In Abbildung 3.9a sind die blau dargestellten Flächen so verknüpft.

Durch diese Verknüpfung lässt sich das Prinzip der assoziativen Modellkopplung insofern erweitern, als dass durch die Anpassung des Parameterwerts der Extrusion, die den mittleren Quader erzeugt, die Position der zusammengesetzten Bauteile automatisch durch das CAD-System angepasst wird. Diese Anpassung ist letztlich sogar zwingend notwendig, da sonst die bereits definierten topologischen Zwangsbedingungen nicht mehr eingehalten wären. Einige CAD-Systeme bieten an dieser Stelle sogar noch weitere Funktionen: Es ist möglich, ein Bauteil als *adaptiv* oder *anpassbar* zu definieren, sodass sich Parameterwerte dieses Bauteils automatisch verändern, wenn dies zur Einhaltung der topologischen Zwangsbedingungen notwendig ist.

Wird also im in Abbildung 3.9 dargestellten Beispiel der mittlere Quader adaptiv geschaltet und anschließend der Abstand zwischen den beiden anderen Quadern auf einen von der Extrusionslänge des mittleren Quaders verschiedenen Wert definiert, so passt sich die Extrusionslänge automatisch an. Hierbei muss aber wie bei der Modellierung von Skizzen durch den Nutzer sichergestellt werden, dass keine Widersprüche entstehen, da der GCS des CAD-Systems sonst keine gültige Lösung berechnen kann. Gerade bei Verwendung

mehrerer Skizzen als Basis von verschiedenen Volumenkörpern und der Anordnung dieser Körper mittels topologischer Zwangsbedingungen, ist viel Erfahrung auf Seiten des Nutzers notwendig. Zusätzlich muss ein leistungsstarker GCS vorhanden sein, der den Nutzer bei der Modellierung insofern unterstützt, als dass Änderungen einerseits schnell auf das gesamte Modell propagiert werden und der Nutzer andererseits direkt auf Widersprüche, die durch eine Modellierungsoperation entstehen können, hingewiesen wird.



(a) **(b)**

Abbildung 3.9: Der mittlere Quader ist ein adaptives Bauteil, dessen Extrusionslänge sich automatisch an einen veränderten Abstand der beiden anderen Quader anpasst.

3.2.2 Struktur und Historie eines parametrischen Modells

In gängigen parametrischen CAD-Anwendungen werden parametrische Modelle in Skizzen, Bauteile und Baugruppen gegliedert. Diese Gliederung wurde auch für die Struktur des in dieser Arbeit entwickelten Ansatzes übernommen, da die Implementierung des Ansatzes für bestehende parametrische CAD-Systeme so leichter möglich wird. Im Folgenden wird daher kurz auf die entsprechenden Begriffe im Allgemeinen eingegangen. Eine ausführliche Beschreibung hinsichtlich der Verwendung im hier entwickelten Ansatz wird im Kapitel 5 vorgenommen.

Ein Kernkonzept der parametrischen Modellierung besteht darin, nicht nur das Endergebnis eines Modellierungsprozesses zu speichern, sondern auch die Abfolge der einzelnen Modellierungsoperationen, die durch einen Anwender in einem CAD-System ausgeführt werden[4]. Die Abfolge dieser Schritte wird als die Konstruktionshistorie eines Modells bezeichnet. Modelle, die auf diese Weise erstellt werden, werden auch prozedurale Modelle oder historienbasierte Modelle genannt (Obergrießer, 2016; Vajna *et al.*, 2018). Die Speicherung der Historie eines Modells ermöglicht es, im Nachhinein Änderungen vorzunehmen, die über die reine Änderung der Werte von Parametern hinausgehen. Beispielsweise kann eine Skizze, die die Basis einer Extrusion bildet, neu definiert werden, sodass sich der durch die Extrusion er-

[4]So wie auch bei der impliziten Repräsentation von Modellen.

zeugte Körper und gegebenenfalls auf diesem Körper aufbauende Modellierungsoperationen entsprechend verändern.

Allerdings ist die Historie der Modellierungsoperationen nicht in allen Teilen eines parametrischen Modells relevant. Grund hierfür ist der bereits beschriebene Unterschied zwischen dem prozedural-parametrischen und dem variationalen Modellierungsansatz. Durch die Kombination dieser beiden Ansätze zu einem hybriden Konzept ergeben sich Modelle, bei denen ein Teil der Modellierungsschritte in einer bestimmten Reihenfolge gespeichert werden muss. Andere Teile des Modells können hingegen nach der Modellierung als explizites Ergebnis gespeichert werden. Diese Unterschiede sind wesentlich für das Verständnis des Ablaufs der parametrischen Modellierung und daher auch für die Beschreibung des Modellierungsprozesses. Insofern wird bei der Beschreibung von Skizzen, Bauteilen und Baugruppen darauf eingegangen.

Wie bereits im vorigen Abschnitt beschrieben, bilden Skizzen[5] in einem parametrischen CAD-System die Grundlage für die Erstellung von dreidimensionalen Volumenkörpern, aus denen ein Modell zusammengesetzt wird. Die Erstellung einer Skizze folgt dem variationalen Modellierungsansatz. Der Anwender kann geometrische Elemente erstellen und grundsätzlich beliebig anordnen oder nach der Erstellung verschieben. Dabei wird in der Regel zuerst ein ungefährer Entwurf der Skizze erstellt, bei dem Abmessungen und Ausrichtung der Elemente noch nicht final sind. Erst durch das Hinzufügen parametrischer Zwangsbedingungen werden die Positionen der geometrischen Elemente tatsächlich festgelegt. Teils werden durch das CAD-System bei der Erstellung der geometrischen Elemente bereits Zwangsbedingungen erstellt, um die wahrscheinliche Entwurfsabsicht des Nutzers direkt abzubilden: Wird beispielsweise eine Linie annähernd horizontal gezeichnet, so wird dem Nutzer vorgeschlagen, diese Linie über eine Zwangsbedingung als horizontal zu definieren. Sobald der Nutzer dimensionale Zwangsbedingungen festgelegt hat, kann die Position der geometrischen Elemente auch über die Parameterwerte dieser Zwangsbedingungen verändert werden.

Wichtig ist, dass innerhalb einer Skizze die Abfolge oder Historie der durchgeführten Operationen nicht als Teil der Skizze gespeichert wird. Dies hat zur Folge, dass Geometrie und Zwangsbedingungen unabhängig von einer Reihenfolge angeordnet, miteinander gekoppelt und berechnet werden können (Shah *et al.*, 1995). Trotzdem hat die Reihenfolge, in der geometrische Elemente platziert und durch Zwangsbedingungen verknüpft werden, einen wesentlichen Einfluss auf die finale Position der geometrischen Elemente. Dies bedingt sich auch dadurch, dass die Lösungsfindung durch den GCS von der Positionierung der geometrischen Elemente durch den Nutzer abhängig ist (siehe Abschnitt 3.2.4). Die Geometrie einer Skizze wird vielmehr als Momentaufnahme der Position der geometrischen Elemente und Zwangsbedingungen gespeichert. Das heißt, dass geometrische Elemente hinsichtlich ihrer Erstellung und ihrer Existenz nicht voneinander abhängig sind: Wird beispielsweise

[5](In den folgenden Kapiteln wird teils auch der englische Begriff *sketch* verwendet.

ein Kreis so gezeichnet, dass sein Mittelpunkt auf einer Linie liegt, kann diese Linie gelöscht oder verschoben werden, ohne dass dies den Kreis beeinflusst (außer eine Zwangsbedingung verhindert dies). Dies hat den Vorteil, dass der Nutzer bei der Modellierung sehr intuitiv vorgehen und Änderungen unkompliziert umsetzen kann. Da die Abfolge der Modellierungsoperationen der Skizze allerdings nicht gespeichert wird, kann der Erstellungsprozess nicht einfach nachvollzogen werden. Zusätzlich muss bei jeder Änderung eine Neubewertung der vollständigen Skizze durch den GCS vorgenommen werden. Dies bedeutet letztlich einen größeren Aufwand bei der Berechnung der Skizzengeometrie durch den GCS (Obergrießer, 2016). Außerdem kann durch die große Flexibilität bei der Erstellung nicht sichergestellt werden, dass tatsächlich eine valide Lösung, die der Entwurfsabsicht des Nutzers entspricht, entsteht (Bettig *et al.*, 2003).

Nach Fertigstellung einer Skizze kann mittels prozeduraler Modellierungsoperationen ein Volumenkörper erzeugt werden, der die 3D-Geometrie eines Bauteils beschreibt. Werden auf Basis einer oder mehrere Skizzen verschiedene Volumenkörper innerhalb eines Bauteils erzeugt, können diese durch Boolesche Operationen miteinander kombiniert werden. Dies erfolgt nach dem gleichen Prinzip wie bei der Erstellung von Volumenkörpern mittels Constructive Solid Geometry. Der wesentliche Unterschied zu CSG besteht aber darin, dass die durch prozedurale und Boolesche Operationen erzeugten Volumenkörper als Basis für neue Skizzen dienen können. Der Grund hierfür ist, dass sich eine Operation direkt auf die Geometrie des Modells und auf mögliche konsekutive Modellierungsoperationen auswirkt: Wird beispielsweise eine Skizze, die ein Rechteck beschreibt extrudiert, so kann auf einer der Seitenflächen des resultierenden Quaders eine neue Skizze erstellt werden. Wird in dieser Skizze ein Kreis gezeichnet und die Skizze wiederum extrudiert, kann der so erzeugte Zylinder vom ursprünglichen Quader abgezogen werden, sodass ein Quader mit einer Öffnung entsteht. Da hier Modellierungsoperationen auf den Ergebnissen vorangegangener Operationen aufbauen, ist bei einer Modellierung auf Bauteilebene die Reihenfolge der Operationen in der Regel nicht vernachlässigbar.

Im beschriebenen Beispiel wäre es nicht möglich, den Kreis zu skizzieren und zu extrudieren, bevor der Quader erzeugt wurde. Da die Reihenfolge der Modellierung eine Rolle spielt, wird sie als Historie des Bauteils im parametrischen Modell gespeichert. Dies ermöglicht es auch nachträgliche Änderungen an bereits durchgeführten Modellierungsoperationen vorzunehmen, die zu einer Neuberechnung der nachfolgenden Modellierungsschritte führen und damit die erzeugte Geometrie verändern. Solche Änderungen können beispielsweise die Anpassung des Parameterwerts einer dimensionalen Zwangsbedingung sein oder die Länge einer Extrusion. Natürlich muss hierbei beachtet werden, dass keine Änderungen durchgeführt werden, die die Ausführung der nachfolgenden Operationen unmöglich machen. Dabei ist es auch möglich, dass eine Änderung zu einer möglichen Ambivalenz der nachfolgenden Operationen führt. Dies ist z. B. dann der Fall, wenn eine Änderung vorgenommen wird, die die Seitenfläche eines Körpers in zwei getrennte Flächen aufteilt (siehe Abbildung 3.10).

Dieses Phänomen wird als *persistent naming problem* bezeichnet (Bidarra *et al.*, 2005; Marcheix *et al.*, 2002) und ist im Kontext parametrischer CAD-Werkzeuge in der Literatur ausführlich dokumentiert (Safdar *et al.*, 2020). Bei der Nutzung eines entsprechenden CAD-Werkzeugs ist es also notwendig, dieses Problem zu kennen und zu berücksichtigen, um unerwünschte Modellierungsergebnisse zu vermeiden. Gängige CAD-Werkzeuge verfügen aber über Funktionen, mit denen Konflikte oder Mehrdeutigkeiten, die durch das *persistent naming problem* entstehen, behoben werden können, wobei jedoch ein manueller Eingriff notwendig ist.

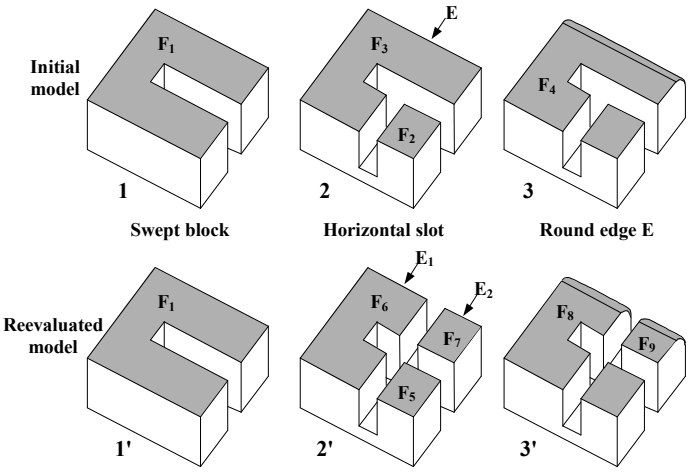


Abbildung 3.10: Grafische Darstellung des *persistent naming problem*. Durch die Veränderung des initialen Modells entstehen neue Flächen und Kanten, sodass nachfolgende Modellierungsoperationen nicht mehr eindeutig definiert sein können. *Quelle: Marcheix* et al.*, 2002.*

Ist die Modellierung eines Bauteils abgeschlossen, kann der so erzeugte Volumenkörper in eine Baugruppe eingefügt werden. Baugruppen können sich aus mehreren Bauteilen zusammensetzen oder auch ineinander verschachtelt sein und so zur Erstellung komplexer geometrischer Modelle genutzt werden. Um die Positionierung der Bauteile in einer Baugruppe festzulegen, werden topologische Zwangsbedingungen genutzt (siehe voriger Abschnitt). Die Platzierung von Bauteilen in einer Baugruppe erfolgt nach dem variationalen Modellierungsansatz, bei dem die Reihenfolge der Modellierungsschritte nur eine untergeordnete Rolle spielt. Änderungen, die an einem Bauteil durchgeführt werden, das Teil einer Baugruppe ist können sich aber auf die Positionierung weiterer Bauteile auswirken, die mittels topologische Zwangsbedingungen von diesem Bauteil abhängig sind. Wie von Obergrießer (2016) beschrieben, werden so bei der Verwendung von Baugruppen in gängigen parametrischen CAD-Werkzeugen die Ansätze der prozedural-parametrischen und der variationalen Modellierung kombiniert.

3.2.3 Parametrische Zwangsbedingungen

In der parametrischen Modellierung werden Zwangsbedingungen eingesetzt, um topologische Abhängigkeiten zwischen geometrischen Elementen in ein Modell zu integrieren. In der deutschen Literatur wird dabei auch oft der im internationalen Kontext übliche englische Begriff *constraint* verwendet.

Die grundlegenden parametrischen Zwangsbedingungen, die von allen wichtigen parametrischen CAD-Anwendungen implementiert werden, wurden von Schultz *et al.* (2015) als *standard geometric constraint language* definiert[6]. Sie umfasst die folgenden geometrischen Zwangsbedingungen: koinzident, kollinear, tangential, horizontal, vertikal, parallel, senkrecht und fixiert. In parametrischen CAD-Systemen wie Siemens NX, Autodesk Inventor oder FreeCAD können teils noch weitere geometrische Zwangsbedingungen verwendet werden: geometrische Elemente können z. B. symmetrisch an einer Achse ausgerichtet werden oder es kann definiert werden, dass zwei Linien die gleiche Länge haben (ohne, dass dafür eine dimensionale Zwangsbedingung genutzt werden muss).

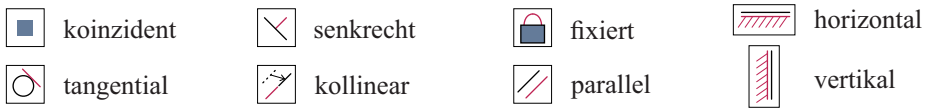


Abbildung 3.11: Parametrische Zwangsbedingungen die nach Schultz *et al.* (2015) in der *standard geometric constraint language* enthalten sind. Es sind nur die geometrischen, nicht aber die dimensionalen Zwangsbedingungen abgebildet.

In der *standard geometric constraint language* sind nach Schultz *et al.* (2015) auch dimensionale Zwangsbedingungen enthalten. Dimensionale Zwangsbedingungen werden eingesetzt, um Abstände zwischen geometrischen Elementen oder die Dimensionen dieser Elemente zu kontrollieren. Es kann sich dabei beispielsweise um horizontale und vertikale Abstände oder um Winkel und Durchmesser handeln. Diese Dimensionen (Abstände, Winkel etc.) werden über Parameter beschrieben, die entweder durch feste Werte oder in Abhängigkeit anderer Parameter definiert werden. Somit können die Parameter direkt voneinander abhängig sein oder in Form von Variablen in mathematischen Formeln aufeinander verweisen. Diese Definition führt dazu, dass durch die Änderung des Werts eines einzelnen Parameters auch alle davon abhängigen Parameter verändert werden und dementsprechend die Beschaffenheit der kompletten Skizze automatisch angepasst wird.

Abbildung 3.11 gibt einen Überblick über die verschiedenen parametrischen Zwangsbedingungen, die in einer Skizze verwendet werden können. Die im Rahmen dieser Arbeit verwendeten parametrischen Zwangsbedingungen werden in Abschnitt 5.5 im Detail beschrieben.

[6] Schultz *et al.* (2015) beziehen sich bei der Definition der *standard geometric constraint language* auf bereits durch Owen (1991) definierte topologische Beziehungen, die eine gute Balance zwischen Anwendbarkeit und Nachvollziehbarkeit bieten.

Eine wesentliche Rolle bei der Erstellung einer Skizze nehmen die verwendeten parametrischen Zwangsbedingungen hinsichtlich der Einordnung der Skizze als unter-bestimmt, über-bestimmt und voll-bestimmt (auch wohlbestimmt) ein. In der Regel wird bei der Erstellung einer Skizze angestrebt, genau so viele parametrische Zwangsbedingungen zu verwenden, dass die Skizze voll-bestimmt ist (Obergrießer, 2016). Dazu müssen alle Freiheitsgrade der verwendeten geometrischen Elemente durch Zwangsbedingungen eingeschränkt oder spezifiziert werden.

Zusätzlich ist es möglich, Bauteile innerhalb einer Baugruppe mit 3D-Zwangsbedingungen zu koppeln, die auch als *Assembly Constraints* bezeichnet werden. Damit kann festgelegt werden, dass ein bereits modelliertes Bauteil relativ zu einem anderen Bauteil positioniert wird. Dies wird in Abschnitt 5.3 detaillierter beschrieben.

3.2.4 Constraint Solver

Im Zuge der Erstellung eines parametrischen Modells wird durch die Definition von Zwangsbedingungen ein System von topologischen Zusammenhängen aufgebaut. Formal betrachtet wird dabei ein Constraint-Problem definiert. Die Lösung dieses Problems beschreibt letztlich die Positionierung der geometrischen Elemente in den 2D-Skizzen des parametrischen Modells und dementsprechend auch die 3D-Geometrie des gesamten finalen Modells. Zur Lösung eines solchen Constraint-Problems werden *Geometrical Constraint Solver* (GCS) verwendet (Anantha *et al.*, 1996; Bettig *et al.*, 2011; Fudos *et al.*, 1997; Owen, 1991).

Geometric Constraint Solver (GCS) sind ein wesentlicher Bestandteil parametrischer CAD-Werkzeuge, ohne den parametrische Funktionalitäten nicht implementiert werden könnten. Während der Modellierung ist die Funktionsweise des Geometric Constraint Solver (GCS) für den Nutzer allerdings in der Regel kaum wahrnehmbar – auch wenn sich die Modellierungsoperationen auf die Lösungsfindung des Geometric Constraint Solver (GCS) genauso auswirken, wie die durch den Geometric Constraint Solver (GCS) ermittelte Lösung auf das Modellierungsergebnis. Entwirft ein Nutzer eine Skizze, wird die Topologie der darin enthaltenen geometrischen Elemente durch die vom Nutzer erstellen Zwangsbedingungen festgelegt. Die Lösung dieses so definierten Systems aus geometrischen Elementen und Zwangsbedingungen erfolgt durch den GCS.

Bei einem Constraint-Problem im Kontext der parametrischen Modellierung handelt es sich um eine spezielle Form des *Constraint Satisfaction Problems* (CSP) (Brüderlin *et al.*, 1998). Das CSP ist als eine Menge von Variablen definiert, deren Werte eine Menge von Randbedingungen oder *Constraints* erfüllen müssen (Kumar, 1992; Meseguer, 1989). Eine Lösung eines solchen Problems besteht nach Kolbe (1999) darin, die Werte aller Variablen mit genau einem Wert eindeutig zu belegen, sodass alle Randbedingungen erfüllt sind. Gibt

es mehrere solcher Lösungen für ein CSP, wird die Menge dieser Lösungen als *Lösungsraum eines CSP* bezeichnet.

Im Kontext der parametrischen Modellierung sind die Variablen die Positionen der geometrischen Elemente und die Constraints die parametrischen Zwangsbedingungen. Eine Lösung des parametrischen Constraint-Problems ist daher die Positionierung aller geometrischen Elemente in einer Skizze, bei der alle Zwangsbedingungen eingehalten werden. Die Positionen der Elemente werden über ihre Koordinaten beschrieben. Werden topologische Zwangsbedingungen zur Ausrichtung von Bauteilen in einer Baugruppe eingesetzt, müssen auch diese Randbedingungen bei der Formulierung des Constraint-Problems berücksichtigt werden. Auch ein parametrisches Constraint-Problem kann einen Lösungsraum haben, sofern es mehr als eine Möglichkeit gibt, die geometrischen Elemente entsprechend der Zwangsbedingungen korrekt anzuordnen. Dieses Phänomen und seine Auswirkung auf den Modellierungsprozess wird im übernächsten Abschnitt beschrieben.

Die Arbeitsweise des GCS ist für den Nutzer eines CAD-Systems in der Regel nicht direkt einseh- und nachvollziehbar. Die Erstellung eines parametrischen Modells, das tatsächlich der Entwurfsabsicht des Nutzers entspricht, ist aber wesentlich davon abhängig, wie der GCS auf das Erstellen von Zwangsbedingungen oder die Änderungen von Parameterwerten reagiert. Insofern muss der Nutzer entweder über sehr viel Erfahrung mit einem CAD-System verfügen oder das Verhalten des GCS in verschiedenen Fällen beobachten, um abschätzen zu können, wie sich Veränderungen der Zwangsbedingungen oder Parameterwerte auf das Modell auswirken. Hier ist es sehr hilfreich, wenn bestimmte Skizzierungs- und Modellierungsoperationen formal so definiert werden können, dass die Entwurfsabsicht berücksichtigt wird. Hier kann die formale Repräsentation parametrischer Modellierungsoperationen durch ein Graphersetzungssystem genutzt werden, da der Nutzer die so repräsentierten Operationen ausführen kann, ohne die darin enthaltenen parametrischen Zwangsbedingungen im Detail kennen zu müssen.

Funktionsweise

In der Literatur wird die Funktionsweise des GCS laut Bettig *et al.* (2011) wie folgt beschrieben:

> *Given a set of geometric objects, such as points, lines and circles; given a set of geometric and dimensional constraints, such as distance, tangency, perpendicularity etc.; and given an ambient space, usually the Euclidean plane; assign coordinates to the geometric objects such that the constraints are satisfied, or report that no such assignment has been found.*

Die endgültige Geometrie einer Skizze (also die finalen Koordinaten der einzelnen geometrischen Elemente) wird erst durch die Berechnungen des GCS festgelegt. Im Rahmen des in dieser Arbeit entwickelten Ansatzes spielen GCS insofern eine Rolle, als dass sie die korrekte Interpretation und Darstellung der graphbasierten Repräsentation von Skizzen und Modellen innerhalb einer CAD-Anwendung beeinflussen. Daher ist es notwendig, prinzipiell nachvollziehen zu können, wie der entsprechende GCS dabei vorgeht.

Die Entstehung von GCS ging stark mit der Entwicklung von parametrischen CAD-Systemen einher. Nachdem diese in den 1970er und 1980er Jahren verfügbar und weiterentwickelt wurden, haben sich auch GCS zu Beginn der 1990er Jahre als standardmäßiges Hilfsmittel zur Erstellung von Skizzen durchgesetzt (Bouma *et al.*, 1995; Owen, 1991). Grundlagen und verschiedene Ansätze zur Entwicklung von GCS werden unter anderem von Brüderlin *et al.* (1998), Hidalgo *et al.* (2012), Hoffmann *et al.* (2002, 2005), Imbach *et al.* (2014) und Joan-arinyo *et al.* (2001) beschrieben. Eine umfangreiche Untersuchung über die Weiterentwicklungen der GCS seit der Jahrtausendwende ist in (Bettig *et al.*, 2011) aufgeführt. Diese kommt einerseits zu dem Schluss, dass sich an der fundamentalen Modellierung von Geometrie, die über Parametrik gesteuert wird, wenig geändert hat. Vielmehr wird die Flexibilität der Systeme dadurch eingeschränkt, dass Beziehungen zwischen Parametern immer nur in eine Richtung möglich sind, dass also Parameter nicht gegenseitig voneinander abhängig sein können. Die Methoden innerhalb eines GCS, die zur Lösung der Constraint-Probleme genutzt werden, sind allerdings im untersuchten Zeitrahmen wesentlich verbessert worden, was auch auf verbesserte Hardware zurückzuführen ist. Eine detaillierte Beschreibung der verschiedenen Ansätze von GCS ist beispielsweise in Obergrießer (2016) zu finden.

Insgesamt konnten sich laut Bettig *et al.* (2011) aber die Solver, die mit einem graphbasierten Ansatz arbeiten, am stärksten durchsetzten. Beim diesem Ansatz erfolgt eine Repräsentation des Constraint-Problems durch einen gelabelten Graphen, der als Constraint-Graphen bezeichnet wird. Die Knoten dieses Graphen repräsentieren dabei die geometrischen Objekte, die mit Zwangsbedingungen versehen werden sollen, während mittels der Kanten die Zwangsbedingungen beschrieben werden. Dabei wird zwischen drei wesentlichen Arten von Ansätzen unterschieden.

- Zur Lösung des Constraint Satisfaction Problems wurde die *lokale Constraint-Propagation* als eine der ersten Methoden entwickelt (Bouma *et al.*, 1995; Obergrießer, 2016). Bei diesem Ansatz werden alle Zwangsbedingungen auf Basis einer einzelnen bereits gelösten Variablen einer Zwangsbedingung (z. B. einer Koordinate eines Punktes) propagierend berechnet. Dieser Ansatz unterliegt jedoch einer Reihe von Einschränkungen; es dürfen beispielsweise keine zyklischen Abhängigkeiten zwischen den Constraints vorliegen.

- Der *konstruktive Ansatz* zerlegt und rekombiniert den Constraint-Graph, um die grundlegenden Konstruktionsschritte, die gelöst werden müssen, zu identifizieren (Fudos *et al.*, 1997). In einer zweiten Phase werden diese Schritte unter Verwendung algebraischer und/oder numerischer Methoden ausgearbeitet.

- Ein weiterer Ansatz besteht in der *Analyse der Freiheitsgrade*. Hier werden die Knoten des Graphen mit der Anzahl der Freiheitsgrade der geometrischen Elemente gelabelt. Die Kanten des Graphen werden mit der Anzahl an Freiheitsgraden beschriftet, die durch die verwendeten Zwangsbedingungen aufgehoben werden. Dieser Graph wird anschließend analysiert um eine Lösungsstrategie zu entwickeln. Latham *et al.* (1996) zerlegen den Graph hierfür z. B. in Subgraphen, die hinsichtlich vordefinierter Muster gelöst werden.

Neben solchen graphbasierten GCS gibt es noch verschiedene weitere Ansätze zur Lösung des Constraint-Problems (Bettig *et al.*, 2011):

- Wird ein *logischer Ansatz* verwendet, so wird das Constraint-Problem in geometrische Aussagen und Axiome übersetzt. Dies wird beispielsweise von Aldefeld (1988), Brüderlin *et al.* (1990) und Verroust *et al.* (1992) beschrieben.

- *Algebraische Ansätze* formulieren auf Basis des Constraint-Problems ein Gleichungssystem, in dem die Koordinaten der geometrischen Elemente die Variablen darstellen. Die parametrischen Zwangsbedingungen werden durch die Gleichungen beschrieben. Der Nachteil dieses Ansatzes liegt darin, dass es sehr aufwändig ist, das Gleichungssystem zur Lösung aufzuteilen und die Lösung des gesamten Problems ineffizient wird (Bettig *et al.*, 2011). Wurde ein Constraint-Problem aber mittels eines anderen Ansatzes hinreichend gegliedert, so können die entstandenen Teile mittels des algebraischen Ansatzes gut gelöst werden.

- Auch bei *numerischen Ansätzen* werden Gleichungssystem zur Lösung des Constraint-Problems verwendet. Hier erfolgt die Lösung iterativ und z. B mit Hilfe des Newtonverfahrens. Falls ein Constraint Problem mehrere Lösungen hat, finden numerische Methoden allerdings teils nur manche dieser Lösungen, sodass der Nutzer möglicherweise keine Möglichkeit hat, die von ihm gewünschte Lösung auszuwählen. Numerische Ansätze wurden unter anderem durch Borning (1981), Hillyard *et al.* (1978) und Light *et al.* (1982) untersucht.

- Weiterhin existieren *symbol-basierte Ansätze* (Chou, 1988) und *Theorem-Proving Ansätze* (Chou, 1988).

Die genannten Ansätze werden nach Obergrießer (2016) in wissenschaftlichen Arbeiten häufig nur separat betrachtet, obwohl in der Praxis Verfahren notwendig sind, die eine

Vielzahl von verschiedenen Constraint-Problemen lösen können. In parametrischen CAD-Werkzeugen sind daher meist GCS wie D-Cubed (Siemens AG, 2022) oder Ledas (Ershov, 2022) implementiert, die verschiedene Ansätze kombinieren (Obergrießer, 2016).

Einfluss von Nutzereingaben auf die Lösungsfindung des GCS

Durch fehlende oder mehrdeutige Zwangsbedingungen sind in vielen Fällen verschiedene Lösungen des Constraint-Problems möglich. In CAD-Anwendungen wird diese Problematik meist durch die Erstellung der Geometrie innerhalb der graphischen Benutzeroberfläche gelöst. Während des Zeichnens der einzelnen geometrischen Elemente wird bereits ungefähr festgelegt, an welcher Stelle in der Skizze diese tatsächlich liegen. Durch diese vorläufige bzw. temporäre Positionierung ergibt sich für den GCS bereits eine annähernd richtige Lösung in Form der vorliegenden Geometrie. Bei der Evaluierung der möglichen Lösungen des Constraint-Problems wird dann die Lösung ausgewählt, die der eingegebenen Geometrie am ähnlichsten ist und insofern mit einer hohen Wahrscheinlichkeit der durch den Nutzer angestrebten Lösung entspricht (Jubierre, 2009). Dieses Verfahren ist zwar mathematisch nicht eindeutig, es liefert jedoch in der Praxis in den meisten Fällen sehr gute Lösungen, selbst wenn sich der Nutzer nicht zwangsläufig über die Funktionsweise des GCS im Klaren ist.

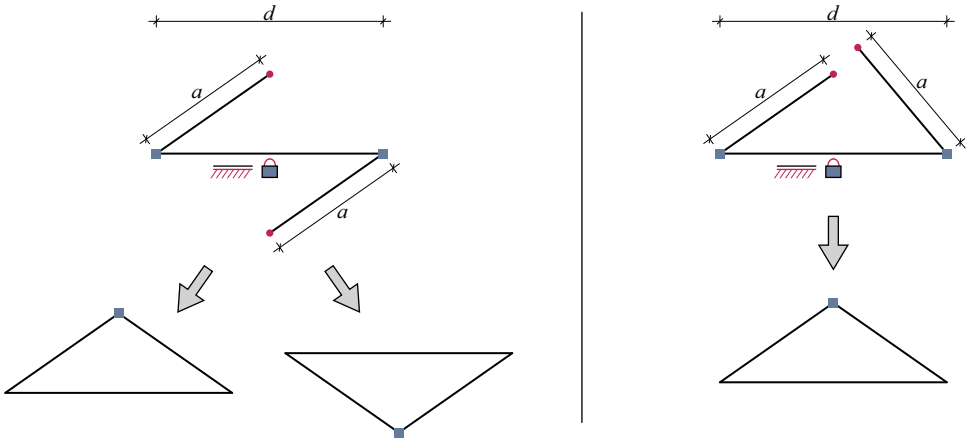


Abbildung 3.12: Ein Constraint-Problem mit zwei möglichen Lösungen. Werden hier die rot markierten Endpunkte koinzident gesetzt sind die beiden darunter dargestellten Resultate eine Valide Lösung (links). Wird bei der Modellierung allerdings wie rechts dargestellt vorgegangen, wird durch einen GCS in der Regel die unterhalb dargestellte Lösung gewählt.

Ein Szenario, bei dem der Lösungsraum des Constraint-Problems zwei mögliche Lösungen enthält, ist in Abbildung 3.12 (links) beschrieben. Hier ist eine Skizze dargestellt, die eine Linie mit einer festen Länge d enthält, die mittels parametrischer Zwangsbedingungen horizontal ausgerichtet und an ihrer Position fixiert ist. An diese Linie schließen zwei weitere Linien mit festen Längen an. Je ein Ende dieser Linien ist als koinzident zu einem der Endpunkte der horizontalen Linie definiert. Im nächsten Modellierungsschritt sollen nun

die freien Enden dieser Linien koinzident gesetzt werden. So ergeben sich die unterhalb dargestellten Resultate, die beide eine valide Lösung des Constraint-Problems sind. Da die ursprüngliche Skizze symmetrisch aufgebaut wurde, ist für den GCS allerdings keine Entwurfsabsicht erkennbar, sodass nicht vorhergesagt werden kann, welche Lösung der CGS berechnet. Wird aber wie auf der rechten Seite der Abbildung 3.12 modelliert, so wählt der GCS im Normallfall die naheliegendere Lösung (unten).

Im Rahmen dieser Arbeit ist diese Problematik insofern relevant, als dass sie bei der Konzeption der graphbasierten Darstellung berücksichtigt werden muss. Bei der Repräsentation parametrischer Skizzen durch einen Graphen müssen also Informationen zur Entwurfsabsicht so hinterlegt werden, dass der GCS eines CAD-Werkzeugs im Fall eines Constraint-Problems mit mehr als einer möglichen Lösung die gewünschte Lösung erzeugt. Diese Thematik und die dafür entwickelte Lösung wird in Kapitel 6 im Detail beschrieben.

Kapitel 4

Graphen und Graphersetzungssysteme

In dieser Arbeit werden Graphen zur Repräsentation von parametrischen Modellen und Graphersetzungsregeln zur Veränderung dieser Repräsentationen eingesetzt. Veränderungen eines solchen Modells können damit durch Graphersetzungsregeln gesteuert werden. So entsteht ein Graphersetzungssystem, mit dem parametrische Modelle auf Basis ihrer graphbasierten Repräsentationen erstellt, verändert und detailliert werden können. Die Graphentheorie bildet dabei die formale Grundlage, auf deren Basis diese Repräsentationen erstellt werden. Die Veränderung der Graphen erfolgt mit einem regelbasierten Graphersetzungssystem.

Im folgenden Kapitel wird auf die Grundlagen der Theorie von Graphen und Graphersetzungssystemen eingegangen. Dabei wird zuerst die Graphentheorie als Teilgebiet der Mathematik vorgestellt und darauf aufbauend der Zusammenhang mit konkreten Anwendungsmöglichkeiten hergestellt. Anschließend wird auf den Aufbau eines Graphen und auf Techniken zur Transformation eines Graphen eingegangen, um so das Konzept von Graphersetzungssystemen zu beschreiben.

Neben den grundlegenden mathematischen und formalen Definitionen, den gängigen Bezeichnungen und beispielhaften Anwendungsmöglichkeiten werden dabei auch verschiedene Möglichkeiten zur computergestützten Speicherung und Visualisierung von Graphen, sowie der Ausführung von Graphersetzungsoperationen vorgestellt. Weiterhin wird auf den Aufbau und den Funktionsumfang verschiedener Software-Frameworks zur Graphersetzung eingegangen.

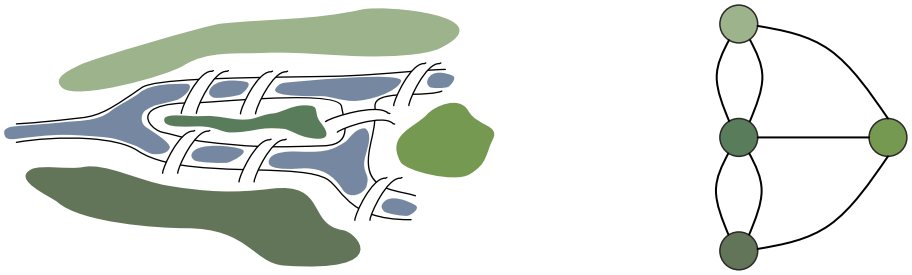
4.1 Grundbegriffe der Graphentheorie

Der Begriff Graphentheorie beschreibt ein Teilgebiet der Mathematik, das sich mit der Untersuchung der Zusammenhänge innerhalb von formal definierbaren Netzstrukturen beschäftigt (Tittmann, 2003). Diese Strukturen werden als Graph bezeichnet und stellen eine Menge von Objekten und deren Beziehungen zueinander dar. Die Objekte werden Knoten genannt. Die in der Regel paarweisen Beziehungen zwischen diesen Knoten heißen Kanten. Graphen dienen als mathematische Modelle, mit denen sich konkrete Fragestellungen aus Physik, Chemie, Informatik, Genetik, Psychologie, Soziologie, Linguistik und auch der Ingenieurwissenschaften analysieren lassen, indem sie als Probleme der Graphentheorie formuliert werden (Balakrishnan *et al.*, 2000; He *et al.*, 2008).

Diesen abstrakten Modellen sind in der grundlegenden mathematischen Definition keine Informationen über die genaue Art oder die Beschaffenheit der Knoten und der Kanten zugeordnet. Trotz dieses sehr hohen Abstraktionsgrades kann eine Vielzahl von Eigenschaften eines Graphen untersucht werden, die sich nur aus der Anordnung seiner Knoten und Kanten ableiten lassen. Als Beispiel sind die Suche nach dem kürzesten Weg zwischen zwei Knoten, die durch Kanten miteinander verbunden sind, oder die Frage, ob ein bestimmter Graph einem Teil eines anderen Graphen entspricht, zu nennen.

Je nach Anwendungsgebiet ist es jedoch auch möglich und oft sinnvoll, die Bestandteile eines Graphen weniger abstrakt zu betrachten. Dazu können verschiedene Typen von Knoten und Kanten definiert werden. Zusätzlich kann festgelegt werden, dass die Knoten und Kanten je nach Typ bestimmte Eigenschaften – die hier als Attribute bezeichnet werden – aufweisen sollen. Durch diese Anreicherung des Graphen mit zusätzlichen Informationen können auch sehr konkrete Fragestellungen abgebildet und untersucht werden, da die grundlegenden mathematischen Zusammenhänge weiterhin gültig sind. Ein Graph wird dabei zu einer formalen Repräsentation der Realität und ist damit letztlich ein Modell entsprechend den in Abschnitt 2.1.1 eingeführten Hauptmerkmalen des Modellbegriffs. Umso mehr Informationen dabei in den Graphen eingebunden werden, umso konkreter bzw. weniger abstrakt wird dieser in seiner Funktion als Modell.

Die Graphentheorie ist ein bereits seit Jahrhunderten untersuchter Teilbereich der Mathematik. Erste Arbeiten in diesem Feld, wie beispielsweise die von Leonhard Euler publizierte Lösung für das Königsberger Brückenproblem, stammen aus dem 18. Jahrhundert. Hier wurde untersucht, ob es einen Rundgang durch die Stadt Königsberg in Preußen gibt, bei dem jede der sieben Brücken über den Fluss Pregel genau einmal überschritten wird. Euler konnte für dieses Problem durch die formale Darstellung der Zusammenhänge in einem Graph (siehe Abbildung 4.1) eine nicht erfüllbare Bedingung angeben und so beweisen, das kein solcher Rundweg existiert (Schubert, 2009).



Vereinfachter Stadtplan
mit Brücken

Abstraktion als Graph

Abbildung 4.1: Das Königsberger Brückenproblem links als Stadtplan und rechts abstrakt als Graph dargestellt. Die Stadtteile sind durch Knoten und die Brücken durch Kanten repräsentiert (Einfärbung der Knoten entsprechend der durch sie repräsentierten Stadtteile).

Die Grundbegriffe der Graphentheorie als Teilgebiet der diskreten Mathematik und der theoretischen Informatik werden in einer Vielzahl von Standardwerken und Lehrbüchern detailliert beschrieben (Balakrishnan *et al.*, 2000; Diestel, 1996; Schubert, 2009; Tittmann, 2003), die auch die Grundlage der in diesem Kapitel aufgeführten Informationen sind. Da das Gebiet der Graphentheorie an dieser Stelle nicht vollumfänglich beschrieben werden kann, werden im Folgenden nur die wesentlichen Grundlagen und die Begriffe, Definitionen und Zusammenhänge, die für den in dieser Arbeit beschriebenen Ansatz relevant sind, eingeführt.

Es ist dabei zu beachten, dass die Begriffe und Notationen, die von den verschiedenen Autoren verwendet werden, nicht konsistent sind. Dies zeigt sich bereits bei den grundlegendsten Begriffen, da die Bestandteile eines Graphen als Knoten oder Ecken bzw. Kanten, Bögen oder Pfeile bezeichnet werden. Dies gilt sowohl für deutsch- als auch für englischsprachige Veröffentlichungen.

Die in dieser Arbeit verwendeten Begriffe und Notationen orientieren sich hauptsächlich am Standardwerk von Diestel (1996), teils aber mit leichten Anpassungen, um insbesondere die Zusammenhänge im Kontext der Graphtransformation möglichst übersichtlich und lesbar darstellen zu können. Weiterhin lässt sich konkret feststellen, dass es gerade bei bestimmten Typen von Graphen (Digraphen, Multigraphen) inkonsistente Definitionen gibt, die sich nicht nur auf die Begrifflichkeiten und die Notation beschränken, sondern tatsächlich unterschiedliche Aussagen über die Eigenschaften von Graphen treffen. Bei den folgenden Definitionen wird jeweils angemerkt, welche unterschiedlichen Auffassungen gängig sind und welche im Kontext dieser Arbeit gelten sollen.

4.1.1 Definitionen

Einfache Graphen

In Abbildung 4.2 ist ein Graph zeichnerisch dargestellt, in dem als Kreis gezeichnete **Knoten** (oder **Ecken**) durch Linien, die **Kanten**, verbunden sind. In diesem Beispiel sind die Kanten x, y, z und noch drei weitere Kanten ohne Bezeichnung vorhanden. Durch diese Kanten werden die Knoten a, b, c, d, e verbunden. Zu Beginn wird hier ein sogenannter einfacher oder schlichter Graph betrachtet. Ein solcher Graph hat eine endlich Menge an Knoten und keine Mehrfachkanten oder Schlingen, die später eingeführt werden.

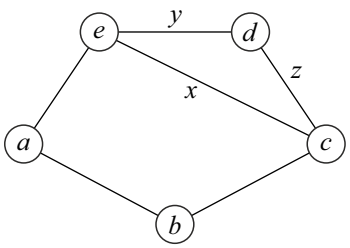


Abbildung 4.2: Beispiel eines Graphen $G = (V, E)$ mit der Knotenmenge $V = \{a, b, c, d, e\}$ und den Kantenmenge $E = \{\{a, b\}, \{b, c\}, \{c, d\} = z, \{d, e\} = y, \{e, c\} = x, \{e, a\}\}$

Ein schlichter Graph G setzt sich demnach aus einer endlichen, nicht leeren Menge V von p Knoten und einer bestimmten Menge $E \subseteq [V]^2$, die also aus q zweiwertigen Teilmengen von V besteht, zusammen. Jedes ungeordnete Paar ($e \in E$ und $e = \{u, v\}$) ist eine Kante von G. Eine Kante e verbindet jeweils u und v und wird allgemein mit $e = \{u, v\}$ beschrieben, es wird aber auch oft die verkürzte Schreibweise $e = uv$ verwendet. Jeder Kante $e \in E$ von G sind damit zwei Knoten aus V zugeordnet.

Allgemein wird der Graph G wie folgt definiert:

- $V = \{v_0, v_1, ..., v_p\}$
- $E = \{e_0, e_1, ..., e_q\}$
- $G = (V, E)$

Diese allgemeine Definition eines Graphen $G = (V, E)$ enthält allerdings noch keine Informationen zur eindeutigen Identifikation von Knoten und Kanten, beispielsweise durch Buchstaben, so wie in Abbildung 4.2 dargestellt. Werden den Knoten und/oder den Kanten eines Graphen Bezeichnungen zugewiesen, so spricht man von einem benannten oder *gelabelten* Graphen. Solche Bezeichnungen werden den Knoten und Kanten in einem benannten Graphen $G = (V, E, f, g)$ über die Abbildungen f bzw. g zugewiesen. f kann dabei beispielsweise eine Abbildung von V in die Menge der natürlichen Zahlen oder die Buchstaben des

Alphabets sein, sodass die Namen der Knoten fortlaufende ganze Zahlen oder Buchstaben sind. Dies gilt analog für Kanten durch die Abbildung g. Damit kann prinzipiell jedem Knoten und jeder Kante eines Graphen ein eindeutiger Identifikator zugewiesen werden, was einerseits die Diskussion über einen bestimmten Graphen stark vereinfacht und andererseits die Definitionen von Abbildungen, Funktionen und Produktionen ermöglicht.

Prinzipiell sind aber auch beliebige andere Namen zulässig, durch die den Knoten auch lesbare Begriffe zugeordnet werden können. In der Literatur wird allerdings in vielen Fällen nicht konkret darauf hingewiesen, dass es sich bei einem Graphen um einen benannten Graphen handelt. Dies ergibt sich dann meist aus dem Kontext, beispielsweise durch eine Darstellung wie in Abbildung 4.2.

Adjazenz und Inzidenz

Wenn zwei Knoten u und v unmittelbar durch eine Kante verbunden sind, dann sind diese Knoten Nachbarn und man nennt sie benachbart oder adjazent. Die Knoten u und v, die durch die Kante e verbunden sind, werden als inzident zur Kante e bezeichnet. Inzidieren verschiedene Kanten denselben Knoten sind diese Kanten benachbart. Paarweise nicht benachbarte Knoten oder Kanten nennt man unabhängig. Im in Abbildung 4.2 dargestellten Graph sind beispielsweise die Knoten d und e bzw. die Kanten e und z benachbart. Die Kante e ist z. B. inzident zum Knoten e aber nicht inzident zum Knoten d. Adjazenz und Inzidenz in Graphen können in Matrizenform über die Adjazenz- und Inzidenzmatrix dargestellt werden.

Kantenfolgen

Eine alternierende Folge von Knoten und Kanten, in der jede Kante inzident zu den in der Folge neben ihr stehenden Knoten ist, wird als *Kantenfolge*[1] bezeichnet. Im obigen Beispiel ist $a, \{a, b\}, b, \{b, c\}, c, x, e, y, d$ eine Kantenfolge im Graphen G. Da alle Knoten in dieser Folge verschieden sind, kann diese Kantenfolge als Weg (siehe Abschnitt 4.1.1) bezeichnet werden. Beginnt und endet eine Kantenfolge am selben Knoten, ist dieser Weg geschlossen und wird als Zyklus bezeichnet. Sind in diesem Zyklus nur Start- und Endknoten identisch, ist diese Kantenfolge ein Kreis. Dies ist bei der Folge c, x, e, y, d, z, c der Fall.

Homomorphismen und Isomorphismen

Bereits mit diesen grundlegenden Definitionen können die für Graphersetzungsoperationen wichtigen Begriffe Graphisomorphismus und -homomorphismus eingeführt werden. Dazu

[1] Einige Autoren verwenden auch den Begriff *Kantenzug*.

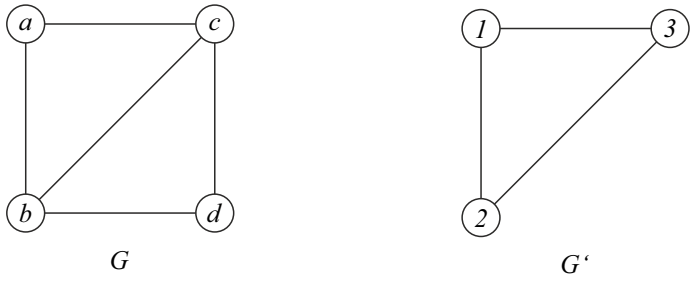


Abbildung 4.3: Kann der Graph G durch α entsprechend (4.1) auf G' abgebildet werden, so ist α ein Homomorphismus.

betrachtete man zwei Graphen $G = (V, E)$ und $G' = (V', E')$. Eine Abbildung $\alpha : V \to V'$ ist ein Homomorphismus von G nach G', wenn jede Kante in G auf eine Kante in G' abgebildet wird, d.h. dass die Nachbarschaftsbeziehungen also die Adjazenz der Knoten erhalten bleiben. Formal ausgedrückt muss also für $\{x, y\} \in E$ immer $\{\alpha(x) = x', \alpha(y) = y'\} \in E'$ gelten. Dadurch gilt im Umkehrschluss auch, dass die in einer Knotenmenge $X \in V$ enthaltenen Knoten, die auf denselben Knoten $x' \in V'$ mit $\alpha(X) = x'$ abgebildet werden, nicht benachbart sind.

Für die in Abbildung 4.3 dargestellten Graphen ist beispielsweise $\alpha : V \to V'$ ein Homomorphismus. α ist dabei wie folgt definiert und es ergibt sich durch die Abbildung der Knoten die rechts dargestellte Abbildung der Kanten:

$$\alpha(X) \quad \text{sei} \quad \begin{cases} \alpha(a) = 1 \\ \alpha(b) = 2 \\ \alpha(c) = 3 \\ \alpha(d) = 1 \end{cases} \qquad (4.1) \qquad \text{Es folgt:} \quad \begin{array}{l} ab \to 12 \\ bc \to 23 \\ cd \to 31 \\ bd \to 21 \\ ac \to 13 \end{array} \qquad (4.2)$$

Aus (4.2) wird direkt ersichtlich, dass alle Kanten in G auf eine Kante in G' abgebildet werden und dass die Knoten a und d, die in G nicht benachbart sind, auf einen Knoten 1 in G' abgebildet werden.

Falls eine Abbildung α sowohl ein Homomorphismus als auch bijektiv ist und auch ihre Umkehrabbildung α^{-1} ein Homomorphismus ist, so ist α ein Isomorphismus. Die Graphen G und G' werden dann isomorph genannt. Dies ist für die Abbildung α, die den in Abbildung 4.3 dargestellten Graphen G auf G' abbildet, nicht der Fall, da sowohl der Knoten a als auch d auf den Knoten 1 abgebildet werden und damit keine Bijektivität gegeben ist. Anders ausgedrückt besteht der wesentliche Unterschied darin, dass ein Homomorphismus die Adjazenz von Knoten erhalten muss, ein Isomorphismus hingegen auch die nicht-Adjazenz von Knoten erhält.

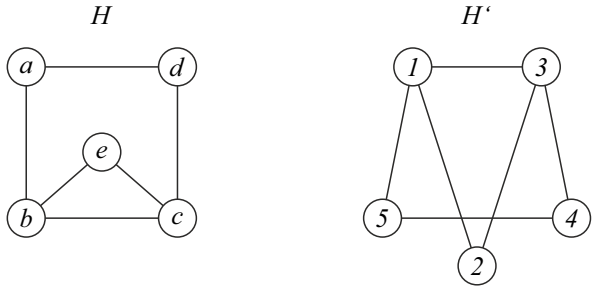


Abbildung 4.4: Zwei isomorphe Graphen H und H'. In 4.3 ist der Isomorphismus β beschrieben.

Die in Abbildung 4.4 dargestellten Graphen H und H' sind hingegen isomorph. Dies zeigt die im Folgenden definierte Abbildung $\beta : H \to H'$, bei der es sich um einen Isomorphismus handelt:

$$\beta(X) \quad \text{sei} \quad \begin{cases} \beta(a) = 5 \\ \beta(b) = 1 \\ \beta(c) = 3 \\ \beta(d) = 4 \\ \beta(e) = 2 \end{cases} \qquad (4.3) \qquad\qquad \text{Es folgt:} \quad \begin{array}{l} ab \to 51 \\ bc \to 13 \\ cd \to 34 \\ da \to 45 \\ be \to 12 \\ ce \to 32 \end{array} \qquad (4.4)$$

Bemerkenswert ist hierbei, dass es für eine bekannte Abbildung eines Graphen auf einen anderen Graphen sehr einfach festzustellen ist, ob diese Abbildung ein Homo- oder Isomorphismus ist. Ist die Abbildung allerdings nicht bekannt, ist die Fragestellung, ob zwei Graphen isomorph sind oder ob ein Homomorphismus existiert, nicht trivial und erfordert bei umfangreicheren Graphen einen nicht unwesentlichen Berechnungsaufwand. Die Komplexität dieses sog. Graphen-Isomorphismus-Problems wird beispielsweise in Garey (1979) ausführlich beschrieben. Es ist dabei insbesondere hervorzuheben, dass kein effizienter (polynomialzeitlicher) Algorithmus zur Prüfung der Isomorphie zweier Graphen bekannt ist (Grohe *et al.*, 2020) und die Komplexitätsklasse des bestmöglichen Algorithmus noch nicht abschließend bestimmt werden konnte. In einer Arbeit von Babai (2015) wurde inzwischen ein Ansatz vorgestellt, der das Graphen-Isomorphismus-Problem in quasipolynomialer Zeit löst. Da das Finden isomorpher Subgraphen im Rahmen der Graphersetzung eine wesentliche Rolle spielt, wird diese Thematik und das entsprechende Vorgehen im Abschnitt 4.2.4 nochmals aufgegriffen.

Teilgraphen

Sollen Graphen mithilfe eines Graphersetzungssystems verändert werden, ist ein Graph in der Regel nicht nur in seiner Gesamtheit zu betrachten. Vielmehr muss untersucht

werden, ob gewisse Muster aus Knoten und Kanten in einem Graphen enthalten sind, also ob der Graph bestimmte Topologien enthält. Im Folgenden werden dazu die Begriffe Teilgraph (Subgraph), Untergraph und Obergraph definiert. Mit diesen Begriffen kann exakt beschrieben werden, in welcher Beziehung zwei Graphen zueinander stehen, wenn ein Graph Bestandteil eines anderen Graphen ist.

Nach Diestel (1996) können dazu die Vereinigungsmenge $G \cup G' := (V \cup V', E \cup E')$ und die Schnittmenge $G \cap G' := (V \cap V', E \cap E')$ zweier Graphen betrachtet werden. Ist die Schnittmenge $G \cap G' = \emptyset$, sind die beiden Graphen disjunkt, sie haben also keine gemeinsamen Knoten (und damit auch keine gemeinsamen Kanten). Unter den Bedingungen $V' \subseteq V$ und $E' \subseteq E$ wird G' als Teilgraph von G bezeichnet und es gilt $G' \subseteq G$. Umgekehrt ist G ein sog. Obergraph (oder Supergraph) von G. Statt Teilgraph wird auch oft der Begriff Subgraph verwendet, der sich auch mit der englischen Bezeichnung deckt. Es kann auch die Formulierung, dass der Graph G den Graphen G' enthält, verwendet werden. G' als Teilgraph von G wird auch als $G[V(G')]$ $G[G']$ bezeichnet.

Anschaulich formuliert bedeutet dies, dass der Subgraph einen Teil (oder alle) der Knoten seines Obergraphen und einen Teil (oder auch alle) der Kanten, die diese Knoten im Obergraphen verbinden, besitzt. In Abbildung 4.5 ist der Graph K mit seinen Subgraphen K' und K'' dargestellt. Hier ist zu bemerken, dass auch K'' ein Subgraph von K ist, obwohl K' zwar alle Knoten, aber nicht alle Kanten zwischen diesen Knoten, die in K existieren enthält.

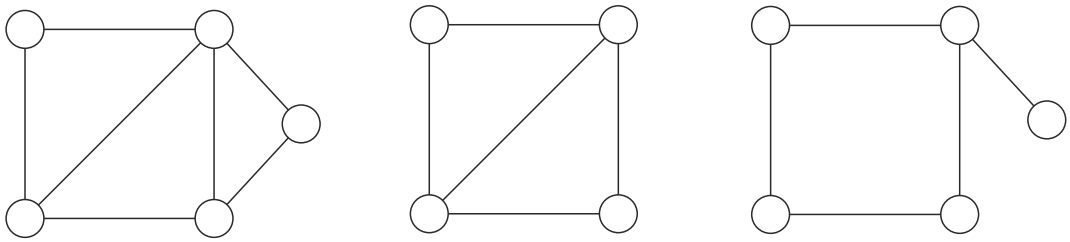


Abbildung 4.5: Die Graphen K' und K'' sind Teilgraphen des Graphen K. Während K' zusätzlich ein Untergraph von K ist, ist dies bei K'' nicht der Fall

Ein spezieller Fall eines Teilgraphen sind sogenannte induzierte Teilgraphen oder Untergraphen. Als induzierter Teilgraph wird ein Teilgraph G' bezeichnet, dessen Knoten durch alle Kanten verbunden sind, durch die die entsprechenden Knoten auch in seinem Obergraph verbunden sind. Formal ausgedrückt müssen dazu alle Kanten $uv \in E(G)$ für deren Endknoten $u, v \in V'(G')$ gilt, in einem Teilgraph enthalten sein. G' wird dann auch mit $G[V']$ bezeichnet. Abbildung 4.5 veranschaulicht die Definition eines induzierten Untergraphen. Der Graph K'' enthält zwar die gleichen Knoten wie K, allerdings sind nicht alle Kanten zwischen diesen Knoten vorhanden.

Ein weiterer spezieller Fall von Teilgraphen sind *Wege* in einem Graphen. Ein Weg ist ein nicht leerer Graph $W = (V, E)$ mit $E = \{x_0, x_1, ..., x_n\}$ und $E = \{x_0x_1, x_1x_2, ..., x_{n-1}x_n\}$. x_0 und x_n sind durch W verbunden und werden als Endknoten von W bezeichnet. Die anderen Knoten sind innere Knoten von W. Die Länge eines Weges wird durch die Anzahl seiner Kanten n bestimmt. Wenn W ein Teilgraph eines Graphen G ist, nennt man W einen Weg in G.

Graphtypen

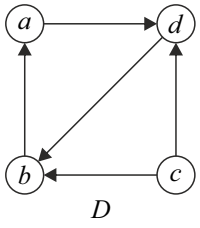
Bisher wurden in diesem Kapitel lediglich einfache Graphen betrachtet. Ein einfacher Graph ist ein vergleichsweise spezieller Typ eines Graphen, da seine mögliche Topologie stark eingeschränkt ist:

- Ein einfacher Graph $G = (V, E)$ muss eine endliche Menge an Knoten haben.
- Eine Kante verbindet immer genau zwei verschiedene Knoten. Es dürfen keine Kanten, die nur inzident zu einem Knoten sind, enthalten sein: $e_1e_1 \notin E$.
- Kanten haben keine Richtung. Es wird also nicht zwischen Start- und Endknoten unterschieden: $v_1v_2 = v_2v_1 \in E$.

Im Folgenden werden die verschiedenen weiteren Typen von Graphen beschrieben.

Gerichtete Graphen

In den bisher besprochenen ungerichteten Graphen haben Kanten keine Richtung, es gilt also $e = \{u, v\} = \{v, u\}$. Sowohl u als auch v sind Endknoten der Kante e (und damit inzident zu dieser Kante). Legt man eindeutig fest, dass in einem Graphen die beiden zu einer Kante inzidenten Knoten, die Start- und Endknoten dieser Kante sind, so wird dieser Graph als gerichteter Graph oder Digraph (von engl. *directed graph*) bezeichnet. In einem gerichteten Graphen werden die Kanten auch als Bögen oder Pfeile bezeichnet und im Gegensatz zu einem gerichteten Graphen mit $e = (u, v)$ bezeichnet, wobei u der Start- und v der Endknoten ist (siehe Abbildung 4.6).



$D = (V_D, E_D)$

$V_D = \{a, b, c, d\}$

$E_D = \{(ba), (ad), (db), (cb), (cd)\}$

Abbildung 4.6: Ein gerichteter Graph und seine formale Definition.

Multigraphen

Multigraphen sind im Vergleich zu den gerichteten Graphen eine weitere Form der Generalisierung. Sie zeichnen sich dadurch aus, dass sie parallele Kanten und Schleifen enthalten können.

Haben zwei Kanten $a = \{u, v\}$ und $b = \{u, v\}$ die gleichen Endknoten spricht man von Mehrfach- oder Multikanten. Die Kanten a und b werden als parallele Kanten bezeichnet. Hat eine Kante $a = \{u, u\}$ denselben Knoten als Start- und als Endknoten, wird diese Kante Schlinge oder Schleife genannt.

Hypergraphen

Eine weitere Variante von Graphen sind sogenannte Hypergraphen. Diese können Hyperkanten enthalten, die nicht nur zwei sondern beliebig viele Knoten verbinden können. Es handelt sich hierbei um die allgemeinste Form eines Graphen, da im Vergleich zu schlichten Graphen, Digraphen und Multigraphen keinerlei Einschränkungen hinsichtlich der Struktur des Graphen vorliegen. Auf die potenzielle Verwendung von Hypergraphen im Kontext dieser Arbeit wird in Kapitel 6 dezidiert eingegangen. Da letztlich keine Hypergraphen verwendet werden, wird hier auf eine vertiefte Beschreibung verzichtet.

Typen und Attribute von Knoten und Kanten

Soll mit einem Graphen eine reale Struktur modelliert werden, reichen Knoten und Kanten, die lediglich eine abstrakte Topologie beschreiben, oft nicht mehr aus, um eine eindeutige und sinnvolle Repräsentation zu ermöglichen, die außerdem alle notwendigen Informationen enthält. Es ist daher sinnvoll, Knoten bzw. Kanten nicht nur als anonyme Bestandteile eines Graphen zu verstehen, die sich nur durch ihre Position im Graph auszeichnen. Wie bereits zu Beginn des Kapitels erwähnt, ist es daher möglich, den Knoten und Kanten eines Graphen verschiedene Typen und Attribute zuzuweisen (Kniemeyer, 2008).

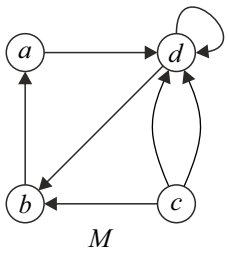


Abbildung 4.7: Ein gerichteter Multigraph oder Multidigraph.

Vom Prinzip her werden dabei, ähnlich wie in der objektorientierten Programmierung, unterschiedliche Typen von Knoten und Kanten definiert, aus deren jeweiligen Instanzen sich der Graph aufbaut. Neben unterschiedlichen Arten von Knoten und Kanten definieren diese Typen gegebenenfalls auch Attribute, also Eigenschaften, die Knoten oder Kanten desselben Typs gemeinsam haben. Diese Attribute kann man als Variablen verstehen, die zwar in ihrer Art jeder Instanz eines Typs gemein sind, sich aber hinsichtlich ihres tatsächlichen Wertes unterscheiden können. Typen und Attribute von Knoten und Kanten eines Graphen können beispielsweise in einem Graph-Metamodell definiert werden. Das entsprechende Vorgehen wird in Abschnitt 4.2.3 beschrieben.

4.1.2 Visualisierung von Graphen

Auch wenn ein Graph rein formal über die Menge seiner Knoten und die Menge der Kanten, die diese Knoten verbinden, beschrieben werden kann, ist es in vielen Fällen hilfreich, einen Graph zu zeichnen. Praktisch zeigt sich dies direkt durch die Abbildungen in den vorangegangenen Abschnitten dieses Kapitels, mit denen die im Text beschriebenen Zusammenhänge visuell dargestellt werden. Sofern der Graph so gezeichnet wird, dass seine topologischen Zusammenhänge schnell erkennbar sind, können die beschriebenen Zusammenhänge veranschaulicht werden. Insofern lässt sich festhalten, dass es sinnvoll ist, Graphen zeichnen zu können, auch wenn dies für die rechnergestützte Speicherung und Verarbeitung von Graphen nicht unbedingt notwendig ist. Dies zeigt sich auch darin, dass praktisch keine Veröffentlichungen im Bereich der Graphentheorie und der Graphersetzung ohne die grafische Darstellung von Graphen auskommen. Auch bei der Definition und Anwendung von Graphersetzungsregeln nimmt das Zeichnen von Graphen eine wichtige Rolle ein, da sowohl für die Erstellung als auch das Verständnis solcher Regeln eine Visualisierung von großem Nutzen sein kann.

Es stellt sich daher die Frage, wie die Visualisierung eines formal definierten Graphen möglichst automatisiert erstellt werden kann, sodass kein aufwändiges manuelles Zeichnen notwendig ist. Im Bereich der Graphentheorie existieren eine Vielzahl von bewährten Algorithmen, mit deren Hilfe Graphen auf verschiedene Arten und damit für verschiedene Anwendungsfälle dargestellt werden können und die teils auch für die Darstellung von sehr großen Graphen genutzt werden können (Zinsmaier *et al.*, 2012). Einen guten Überblick über verschiedene Methoden zum Zeichnen eines Graphen gibt beispielsweise Tamassia (2000).

In dieser Arbeit wird für die Visualisierung der durch das Graphersetzungssystem erzeugten Graphen in erster Linie die Methode des hierarchischen Zeichnens angewendet. Dieses Verfahren zielt darauf ab, in einem gerichteten Graphen eine Hierarchie der Knoten zu identifizieren und den Graphen entsprechend darzustellen. Durch Einsatz dieser Methode

können mit überschaubarem Aufwand gute Ergebnisse erzielt werden, bei denen eine übersichtliche Darstellung der in Kapitel 6 verwendeten Graphen gewährleistet ist.

Neben der Wahl einer bestimmten Anordnung von Knoten und Kanten in der Visualisierung eines Graphen kann die Übersichtlichkeit der Darstellung auch durch die Gruppierung von Subgraphen verbessert werden. Dieses Prinzip wird bei den Graphen, die in den Kapiteln 6 und 7 dargestellt sind, vielfach verwendet. Die folgende Abbildung 4.8 verdeutlicht das entsprechende Vorgehen und wurde mittels desselben Werkzeugs, mit dem die Graphen in den genannten Kapiteln gezeichnet wurden, erzeugt.

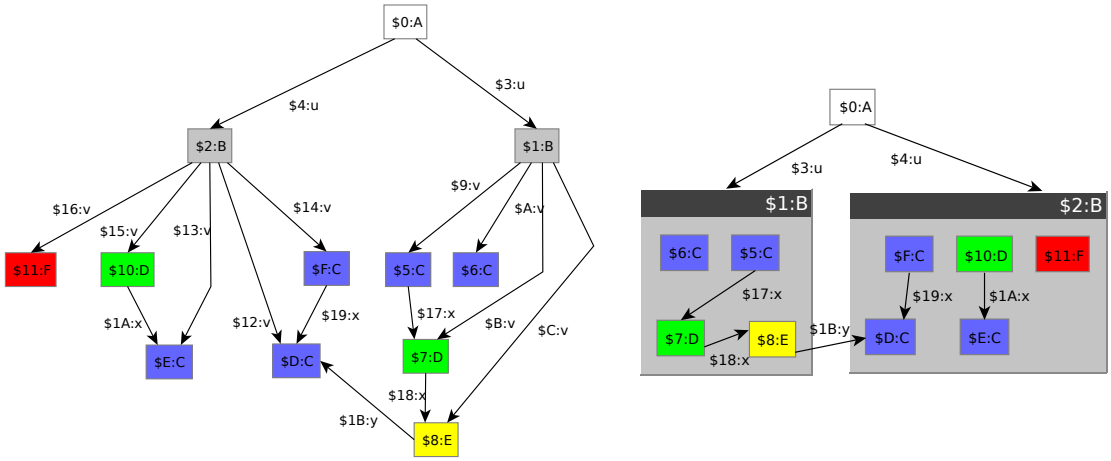


Abbildung 4.8: Zwei verschiedene Darstellungen des gleichen Graphen: Links ist der Graph mit allen Knoten und Kanten dargestellt. Rechts sind die Knoten, die mit v-Kanten mit den B-Knoten verbunden sind, als Gruppe innerhalb dieses Knotens dargestellt.

4.2 Graphersetzung

Als Graphersetzung (engl. *graph rewriting*) wird ein Bereich der Graphentheorie und auch der Informatik bezeichnet, der die regelbasierte Beschreibung der Veränderung oder Transformation von Graphen untersucht. Seltener und vor allem in älteren Veröffentlichungen wird im Deutschen auch der Begriff Graphtransformation verwendet (Frucht, 1939), wobei *graph transformation* im Englischen eine gängige Alternative ist. Generell werden in diesem Abschnitt auch die englischen Begriffe genannt, da diese oft präziserer definiert sind und wissenschaftliche Arbeiten vorrangig in englischer Sprache vorliegen.

Betrachtet man die umfangreiche Literatur im Gebiet der Graphersetzung, lässt sich allerdings feststellen, dass diese Begriffe nicht absolut definiert sind und von verschiedenen Autoren unterschiedlich genutzt werden. Insbesondere im Englischen werden die Begriffe *graph rewriting* und *graph transformation* weitgehend synonym verwendet und nicht dediziert voneinander abgegrenzt (Balasubramanian *et al.*, 2006; König *et al.*, 2018; Machado *et al.*, 2015). Andererseits beschreibt *graph transformation* nach Heckel *et al.* (2020) einen

Teilbereich des *rewriting* von Strukturen und Jakumeit *et al.* (2021) stellen fest, dass die Begriffe teils auch als nicht synonym angesehen werden. In der vorliegenden Arbeit wird in Anlehnung an Jakumeit *et al.* (2021) konsequent der Begriff Graphersetzung verwendet und damit *graph rewriting* und *graph transformation* gleichgesetzt.

Allgemeingültig ist jedoch die Definition der Graphersetzung als die schrittweise Ersetzung von Teilgraphen in einem Obergraphen, den man auch Ausgangs- oder Arbeitsgraph (engl. *host graph* oder *pre-state*) nennt. In der Graphersetzung wird die formale Beschreibung einer bestimmten Veränderung eines Graphen als Graphersetzungsregel bezeichnet. Hier wird auch der Begriff Transformation genutzt, um die Veränderungen, die mithilfe von Graphersetzungsregeln an einem Graphen vorgenommen werden, zu beschreiben: Durch die Anwendung einer Graphersetzungsregel auf einen Arbeitsgraphen, wird dieser transformiert. Das Ergebnis einer solchen Transformation wird als Ziel- oder als modifizierter Arbeitsgraph bezeichnet (engl. *result graph* oder *post-state*).

Abbildung 4.9 zeigt beispielhaft eine einfache Graphersetzungsregel, ihre Anwendung auf einen Arbeitsgraphen und den Zielgraphen nach Ausführung der durch die Regel definierten Graphtransformation.

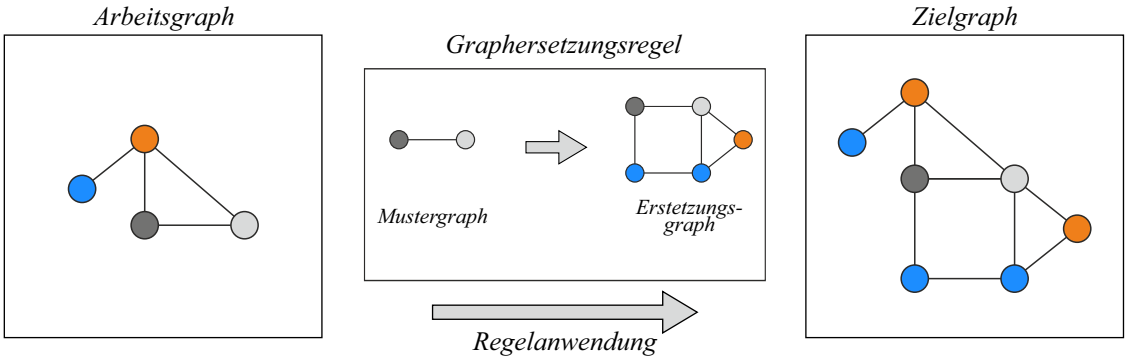


Abbildung 4.9: Beispielhafte Darstellung einer Graphersetzungsregel mit der ein Arbeitsgraph in einen Zielgraphen transformiert wird.

Eine Graphersetzungsregel besteht aus einem Mustergraphen (engl. *pattern graph*) und einem Ersetzungsgraphen (engl. *rewrite graph*). Eine solche Regel kann allerdings nur auf einen Arbeitsgraph angewendet werden, wenn ein Teilgraph dieses Graphen dem Mustergraphen der Graphersetzungsregel entspricht. Es muss also ein Teilgraph des Arbeitsgraphen existieren, der isomorph zum Mustergraphen der Graphersetzungsregel ist. Zu bestimmen, ob ein solcher Isomorphismus vom Mustergraphen der Graphersetzungsregel auf einen Teilgraphen des Arbeitsgraphen existiert, ist ein wesentlicher Schritt einer Graphersetzung und wird in Abschnitt 4.2.4 detailliert beschrieben. Anschließend wird der Ersetzungsgraph in den Arbeitsgraph eingefügt, indem der gefundene Teilgraph so transformiert wird, dass an seiner Stelle ein zum Ersetzungsgraphen isomorphes Abbild entsteht. Wie genau dieses

Einfügen vorgenommen wird, richtet sich nach dem verwendeten Ersetzungsverfahren, siehe Abschnitt 4.2.5.

Eine Menge an Graphersetzungsregeln heißt Graphersetzungssystem. Werden alle möglichen Varianten der aufeinander folgenden Anwendung von Graphersetzungsregeln in einem Graphersetzungssystem betrachtet, spricht man von einer Graphgrammatik (engl. *graph grammar*).

Das Themengebiet der Graphersetzung ist Gegenstand einer Vielzahl von wissenschaftlichen Arbeiten. Die Entwicklungen reichen bis in die frühe zweite Hälfte des 20. Jahrhunderts zurück. Gegenstand der Ansätze war es, die Einschränkungen von linearen Ersetzungssystemen wie Semi-Thue-Systemen zur Stringersetzung (Boone, 1959) und die durch sie definierten Chomsky-Grammatiken (Chomsky, 1956, 1959) auf nicht-lineare Strukturen zu erweitern (Heckel, 2006). Das übergeordnete Ziel dieser Arbeiten, in denen noch der Begriff „web grammar“ verwendet wurde (Pfaltz *et al.*, 1969) waren beispielsweise die Restrukturierung von Algorithmen (Pratt, 1971). In den folgenden Jahren wurden diese Ansätze verfeinert und verschiedene Methoden zur Graphersetzung formal definiert. Einen umfangreichen theoretischen Einblick in die gängigen Ansätze zur Graphersetzung bietet das Standardwerk von Rozenberg (1997), während beispielsweise Heckel *et al.* (2020) eine Einführung in die praktische Anwendung gibt.

4.2.1 Überblick

Das Thema der Graphersetzung lässt sich gut anhand von Wort- oder Stringersetzungssystemen einführen, da die Graphersetzung eine Generalisierung dieses Bereichs darstellt. Stringersetzungssysteme werden daher von verschiedenen Autoren als Analogie genutzt (Kniemeyer, 2008; Plump, 1998). Vereinfacht ausgedrückt besteht ein Stringersetzungssystem aus Substitutionsregeln, die bestimmte Zeichenfolgen entsprechend ihrer Definition verändern. Wird beispielsweise eine Regel $p : X \to Z$ auf die Zeichenfolge XYZ angewandt, so ergibt sich die Zeichenfolge ZYZ (siehe auch Abbildung 4.10). Natürlich ist es auch möglich, mehr als ein Zeichen durch eine Regel zu verändern. Wendet man die Regel $p' : YZ \to YXZX$ auf das Ergebnis der vorigen Operation an, so erhält man die Zeichenfolge $ZYXZX$ auf die sich wieder die erste Regel p anwenden ließe.

Eine Graphersetzungsregel arbeitet weitgehend nach dem gleichen Prinzip, das heißt sie ersetzt eine bestimmte Struktur in einem Graphen durch eine andere Struktur. Wird eine solche Regel auf einen Graphen angewendet, ist es somit möglich, dem Graphen neue Knoten und Kanten hinzuzufügen und bereits bestehende Knoten und Kanten zu entfernen oder durch andere Knoten zu ersetzen. Im Rahmen komplexer Anwendungsszenarien können auch die Attribute von Knoten und Kanten verändern werden, wenn dies durch die Graphersetzungsregeln definiert wird. Generell ist hier aber anzumerken, dass Graphersetzungsregeln

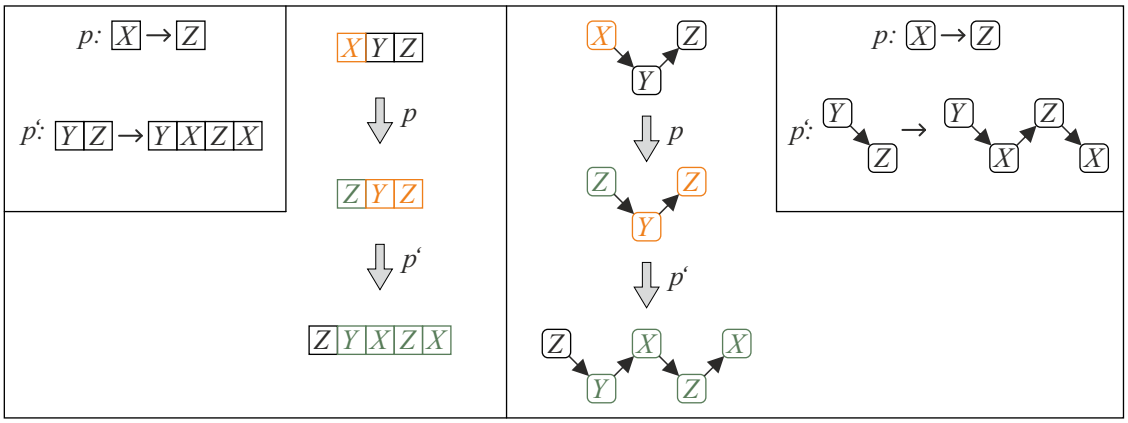


Abbildung 4.10: Die Ersetzungsregeln p und p' und ihre Anwendung sind hier für ein Stringersetzungssystem (links) und ein einfaches Graphersetzungssystem (rechts) dargestellt.

für jede Art von Graph definiert werden können, also beispielsweise sowohl für gerichtete und für nicht gerichtete Graphen.

Im Gegensatz zur Stringersetzung sind bei Graphersetzungsoperationen aber zusätzliche Festlegungen notwendig, um eindeutig zu definieren, wie genau eine Operation durchgeführt bzw. wie eine Graphersetzungsregel angewendet wird. Dies ist in Abbildung 4.10 veranschaulicht. Hier ist links das zu Anfang dieses Abschnitts beschriebene Stringersetzungssystem und die Anwendung der Regeln auf die Ausgangszeichenfolge dargestellt. Die rechte Seite der Abbildung zeigt ein inhaltlich gleiches System, das als gerichteter Graph modelliert wurde. Die Reihenfolge der Zeichen ist hier allerdings nicht implizit durch ihre Anordnung definiert, sondern durch die gerichteten Kanten des Graphen festgelegt, bei dem es sich um einen *Weg* (siehe Abschnitt 4.1.1, S. 82) handelt. Beim Ersetzungsvorgang gilt hier, dass der erste und der letzte Knoten der rechten bzw. linken Seite der Ersetzungsregel einander entsprechen. Nur so ist sichergestellt, dass bei der Anwendung von p' der nicht betroffene Knoten Z (in schwarz) auch nach der Ersetzung adjazent zum ersten Knoten Y des eingefügten Graphen ist.

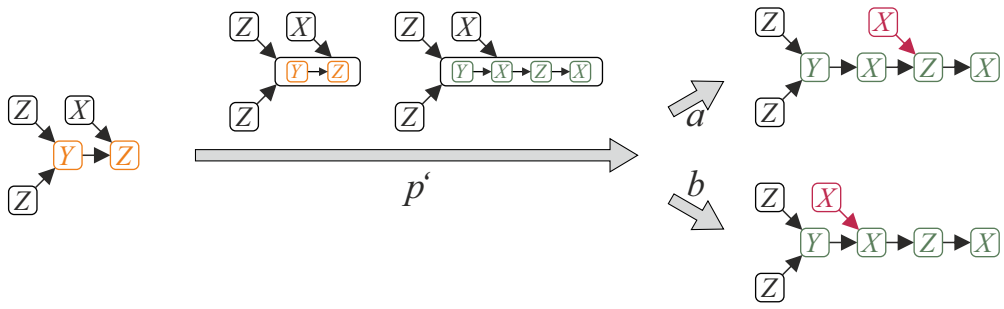


Abbildung 4.11: Anwendung der Graphersetzungsregel p' aus Abbildung 4.10 auf einen komplexeren Arbeitsgraphen. Ohne genaue Definitionen, wie die Ersetzung durchgeführt werden soll, ist das Ergebnis mehrdeutig. Es sind somit beispielsweise die Varianten a und b als Ergebnis möglich.

Abbildung 4.11 verdeutlicht die Notwendigkeit solcher Definitionen bei der Graphersetzung. Hier wird die bereits bekannte Graphersetzungsregel p' auf einen im Vergleich zum vorigen Beispiel komplexeren Arbeitsgraphen angewendet, bei dem es sich nicht mehr um einen *Weg* handelt. Oberhalb des Pfeils, der die Anwendung der Regel kennzeichnet, ist hier der Ersetzungsvorgang detailliert dargestellt: Zuerst wird der Mustergraph $Y \to Z$ gefunden und anschließend durch den Ersetzungsgraphen $Y \to X \to Z \to X$ ersetzt. Die Verbindung dieser Subgraphen zum übergeordneten Graphen ist hier allerdings nicht näher definiert. Dadurch ergeben sich unter anderem die beiden Varianten a und b als Ergebnis der Ersetzung. Variante a als definitives Ergebnis ließe sich hier definieren, indem festgelegt wird, dass der Knoten X auch nach der Ersetzung adjazent zu einem Z-Knoten sein muss. Alternativ wäre Variante b immer das Resultat, wenn Knoten X adjazent zum zweiten Knoten in einem Ersetzungsgraphen, bei dem es sich um einen *Weg* handelt, sein muss. Solche Festlegungen sind natürlich nur dann umsetzbar, wenn auch im Ersetzungsgraphen ein Z-Knoten vorhanden ist (Variante a) bzw. wenn sowohl der Muster- als auch der Ersetzungsgraph ein *Weg* sind (Variante b).

Ein wesentlicher Aspekt der Anwendung einer Graphersetzungsregel besteht dementsprechend darin formal zu definieren, wie die Ersetzung durchgeführt wird. Kniemeyer (2008) setzt für eine solche Formalisierung die folgenden drei Festlegungen voraus:

- Die *Struktur des Graphen*, auf den Ersetzungsregeln angewendet werden sollen, muss definiert sein. Mit Struktur ist beispielsweise gemeint, ob ein Graph gerichtet ist und parallele Kanten (Multikanten) sowie Schleifen erlaubt sind oder nicht.

- Die Festlegung, wie eine *Passung des Mustergraphen* im Arbeitsgraphen definiert ist. Beispielsweise kann es sich bei der Passung um einen Isomorphismus oder einen Homomorphismus handeln.

- Der konkrete *Vorgang der Ersetzung des Mustergraphen* durch den Ersetzungsgraphen und das *Einbinden in den Arbeitsgraphen*. Dies betrifft genau die beschriebene Vermeidung von verschiedenen Varianten als Ergebnis der Ersetzung. Die Eindeutigkeit kann beispielsweise über einen Erhaltungsmorphismus sichergestellt werden, hängt aber letztlich vom Verfahren ab, das zur Ersetzung verwendet wird.

Hieraus lässt sich ableiten, dass das Vorgehen bei der Definition von Graphersetzungssystemen in verschiedener Hinsicht umfänglich definiert sein muss. Insbesondere für das Einbinden des Ersetzungsgraphen in den Arbeitsgraphen wurden hierzu verschiedene Verfahren entwickelt, die beispielsweise in Rozenberg (1997) vollumfänglich beschrieben werden. Im weiteren Verlauf dieses Kapitels werden einige wesentliche Verfahren kurz vorgestellt, nachdem im folgenden Abschnitt zuvor Graphersetzungsregeln näher erläutert werden.

4.2.2 Graphersetzungsregeln

Mit Hilfe von Graphersetzungsregeln lässt sich die Transformation eines Graphen generalisiert beschreiben (Heckel, 2006). Zur Definition einer Graphersetzungsregel muss die zu generalisierende Veränderung im Hinblick auf den Ausgangs- und den gewünschten Zielzustand des Graphen untersucht werden. Dazu wird einerseits der für die Ersetzung relevante Teil des Graphen im Ausgangszustand (ein Subgraph des Ausgangs- bzw. Arbeitsgraphen) und andererseits seine Veränderungen im Zielzustand untersucht. Anschließend kann der Unterschied bzw. die Veränderung zwischen diesen Graphen betrachtet werden.

Eine solche Produktion, die einen Teil des Arbeitsgraphen durch einen neuen Subgraphen ersetzt bzw. diesen Subgraphen an den Arbeitsgraphen anhängt, wird als Graphersetzungsregel bezeichnet. Eine solche Graphersetzungsregel (L, D, R) besteht dabei aus einer linken Seite L (dem *pattern graph* oder Mustergraphen), einer rechten Seite R (dem *rewrite graph* oder Ersetzungsgraphen) und einem Einbettungsmechanismus D (siehe Abschnitt 4.2.5).

Der generelle Ablauf einer Graphersetzungsoperation kann nach Heckel (2006) vereinfacht in den folgenden drei Schritten beschrieben werden:

1. Der Arbeitsgraph H wird nach dem Vorkommen des Mustergraphen L der Graphersetzungsregel durchsucht.

2. Wird ein Vorkommen des Mustergraphen gefunden, wird überprüft, ob eine gegebenenfalls im Mustergraphen definierte negative Anwendungsbedingung[2] zutrifft. Ist dies der Fall wird die Graphersetzungsregel nicht ausgeführt.

3. Es werden alle Knoten und Kanten in H, die mit L übereinstimmen und nicht im R enthalten sind (formal $L \backslash R$), gelöscht. Das Ergebnis dieser Operation wird mit H^- bezeichnet.

4. Alle Knoten und Kanten aus R, die nicht in L enthalten sind (formal $R \backslash L$), werden H^- hinzugefügt. Es entsteht der modifizierte Arbeitsgraph H' als Ergebnis der Graphtransformation.

Im Folgenden wird der sogenannte Single Pushout Approach[3] (SPO) weiter betrachtet. Dieses Vorgehen zur Graphersetzung ist in der folgenden Abbildung 4.12 schematisch dargestellt. Weitere Ansätze zur Graphersetzung werden in Abschnitt 4.2.5 kurz erläutert.

[2] Eine negative Anwendungsbedingung (engl. *negative application condition*) definiert Bedingungen, die im Mustergraphen L nicht zutreffen dürfen (Habel *et al.*, 1996). Somit kann der Kontext, in dem eine Graphersetzungsregel angewendet werden darf, genauer beschrieben werden. Damit ist es möglich, auch einen negativen Kontext zu beschreiben, da es oft erheblich einfacher ist, auszudrücken, dass es gewisse Knoten bzw. Kanten nicht geben darf, als alle gültigen Kontexte zu benennen (Geiß, 2008).

[3] Das in dieser Arbeit entwickelte Graphersetzungssystem und die hierzu genutzte Software zur Graphersetzung GrGen.NET nutzen ausschließlich den SPO (wobei GrGen.NET auch die Graphersetzung

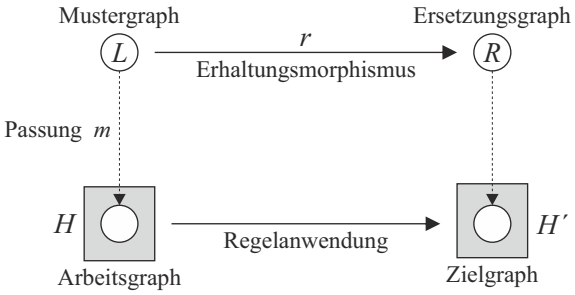


Abbildung 4.12: Grundlegendes Prinzip der Graphersetzung mittels Single Pushout Approach. *Eigene Darstellung angelehnt an Geiß (2008) und Jakumeit* et al. *(2021).*

In dieser Abbildung 4.12 ist neben den bereits beschriebenen Bestandteilen einer Graphersetzungsregel ein sogenannter Erhaltungsmorphismus[4] r zur Einbettung des Ersetzungsgraphen in den Arbeitsgraphen dargestellt. Dieser kennzeichnet die Elemente des Graphen (Knoten oder Kanten) von L und R, die während der Graphtransformation erhalten bleiben sollen (r bildet damit einen Subgraphen in L auf einen Subgraphen in R ab; man spricht von einem partiellen Homomorphismus). Dies kann in einer Graphersetzungsregel beispielsweise dadurch umgesetzt werden, dass die Knoten und Kanten, die erhalten bleiben sollen, in L und in R mit der gleichen Bezeichnung (oder dem gleichen Label) versehen werden (diese Methode wurde in den Graphersetzungsregeln, die in Abbildung 4.14 dargestellt sind, verwendet).

Bei Anwendung des SPO wird im Arbeitsgraphen H nach einem Vorkommen des Mustergraphen L gesucht. Dieses Vorkommen wird als Passungsmorphismus oder Passung m (*match*) bezeichnet und ist graphentheoretisch gesehen ein Graphhomomorphismus, also eine Abbildung von L auf einen Subgraphen in H, bei der die Adjazenz von Knoten erhalten bleibt[5]. Anschließend wird dieses Vorkommen $m(L)$ im Arbeitsgraphen so verändert, dass ein isomorpher Subgraph des Ersetzungsgraphen R entsteht. Bei dieser Veränderung werden Elemente aus L, die in r nicht definiert sind, aus $m(L)$ gelöscht, während Elemente aus R, die nicht in r enthalten sind, H hinzugefügt werden. Alle weiteren Elemente in r werden beibehalten. Dadurch entsteht nach Abschluss des Transformationsvorgangs der Zielgraph H'.

Zur Veranschaulichung dieser Zusammenhänge ist diese allgemeine Beschreibung in Abbildung 4.13 an einem konkreten Beispiel dargestellt. Die Benennung der Graphen in Abbildung 4.13 ist analog zur Definition im vorigen Abschnitt gewählt. Der Erhaltungsmorphismus ist in dieser Abbildung über die beiden roten mit r gekennzeichneten Pfeile definiert. Der

mittels des Double Pushout Approach (DPO) unterstützt). Eine dezidierte Begründung der Entscheidung zur Nutzung des SPO wird im Anschluss an die Erläuterung der weiteren Ansätze gegeben.

[4] engl. *preservation morphism*

[5] Ist m ein Isomorphismus, so muss in H ein Subgraph existieren, der die gleiche Struktur wie L hat. Dies muss bei der Definition einer Graphersetzungsregel definiert werden.

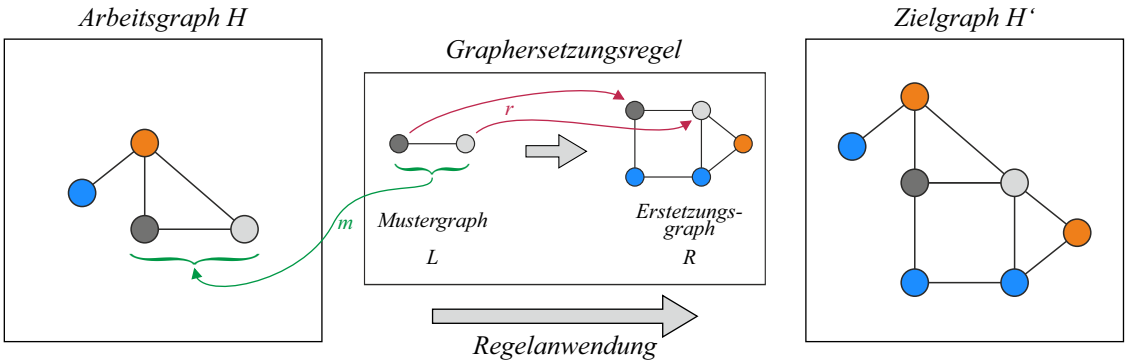


Abbildung 4.13: Die mittig dargestellte SPO-Graphersetzungsregel wird auf den Arbeitsgraphen (links) unter Berücksichtigung des Erhaltungsmorphismus r (rot) angewendet. Die Passung m (grün) definiert, welcher Subgraph des Arbeitsgraphen durch die Ersetzung verändert wird.

Passungsmorphismus m, also der zum Mustergraphen L isomorphe Subgraph in H ist mittels des grünen Pfeils hervorgehoben.

In den Abbildungen 4.14 bis 4.17 sind nun zwei verschiedene Graphersetzungsregeln, *Regel 1* und *Regel 2*, sowie ihre sukzessive Anwendung auf einen initialen Graphen dargestellt. Bei den hier betrachteten Graphen können die Knoten über Label in Form einer Einfärbung unterschieden werden. In Abbildung 4.14 sind hierzu zuerst die beiden Graphersetzungsregeln visuell[6] definiert. Hierbei ist zu beachten, dass im Mustergraph jeweils auch eine negative Anwendungsbedingung enthalten ist, durch die eine Ausführung der Regel verhindert wird. Der Erhaltungsmorphismus r ist über die Kennzeichnung der jeweiligen Knoten durch Buchstaben definiert: In *Regel 1* ist a' auf der rechten Seite äquivalent zu a auf der linken Seite[7]. Da in beiden Beispielen zu allen Knoten im Mustergraphen ein äquivalenter Knoten im Ersetzungsgraphen vorhanden ist, werden durch den Ersetzungsvorgang keine Knoten gelöscht, sondern nur zusätzliche Knoten hinzugefügt.

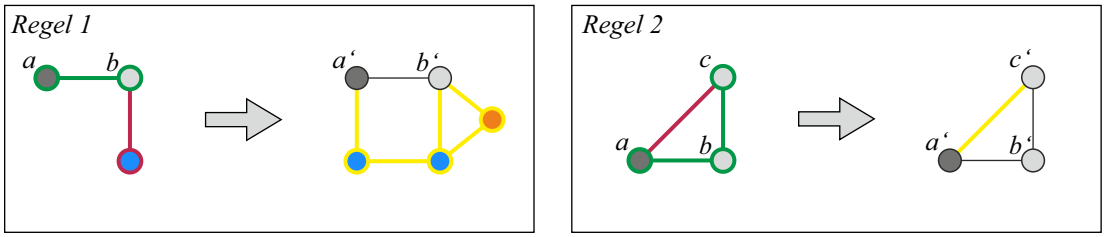


Abbildung 4.14: Definition der Graphersetzungsregeln *Regel 1* und *Regel 2*. Der Mustergraph L ist dabei jeweils in grün dargestellt, die negative Anwendungsbedingung in rot. Im Ersetzungsgraphen R sind die Knoten, die neu hinzugefügt werden, gelb markiert.

In Abbildung 4.15 wird *Regel 1* nun auf einen initalen Graphen angewendet, der somit zu einem Arbeitsgraphen im Kontext der Graphersetzung wird. Dabei wird zuerst der Muster-

[6] Eine Graphersetzungsregel kann aber genauso über die rein formale Beschreibung der Graphen und des Erhaltungsmorphismus definiert werden.

[7] Formal: $r(a) = a'$

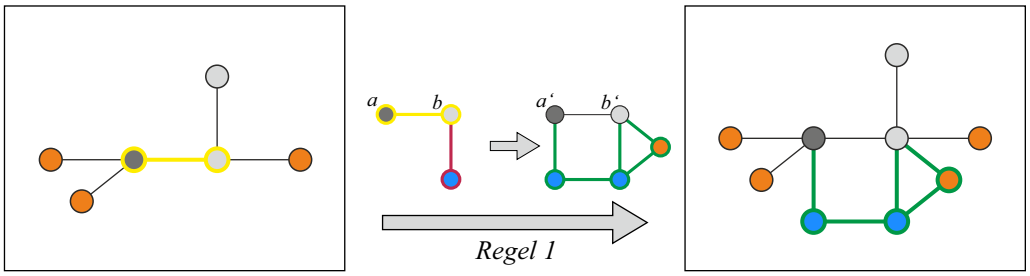


Abbildung 4.15: Anwendung der *Regel 1* auf einen initialen Graphen.

graph (gelb hinterlegt) in diesem Arbeitsgraphen identifiziert. Zusätzlich wird geprüft, ob der identifizierte Subgraph durch das Vorhandensein einer negative Anwendungsbedingung nicht verwendet werden kann. Da aber im Arbeitsgraph kein blau eingefärbter Knoten entsprechend der rot hinterlegten negativen Anwendungsbedingung in *Regel 1* vorhanden ist, kann die Graphersetzungsoperation durchgeführt werden. Daher werden die beiden blauen und der orange Knoten an den Arbeitsgraphen angehängt. Die hierfür neu hinzugefügten Kanten sind auf der rechten Seite der Abbildung im Zielgraphen grün dargestellt.

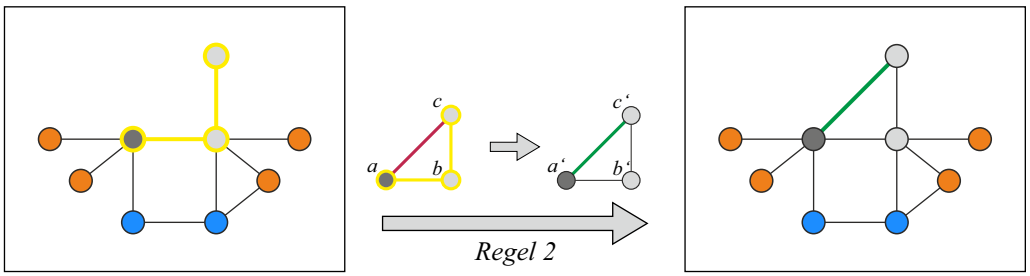


Abbildung 4.16: Anwendung der *Regel 2* auf den Zielgraphen aus Abbildung 4.15.

Dieser Zielgraph dient nun als Arbeitsgraph für eine weitere Graphersetzungsoperation. In Abbildung 4.16 ist dargestellt, wie *Regel 2* auf diesen Arbeitsgraphen angewendet wird. Da der durch *Regel 2* definierte Mustergraph in diesem Arbeitsgraphen vorhanden ist und diese Passung die negative Anwendungsbedingung in Form der Kante $\{a, c\}$ nicht beinhaltet, ist die Ausführung der Graphersetzungsoperation möglich. Somit wird die Kante $\{a', c'\}$ in den Graphen eingefügt. Hierbei ist zu beachten, dass die Anwendung der Regeln *1* und *2* im hier dargestellten Beispiel kommutativ ist und es damit auch möglich wäre zuerst *Regel 2* anzuwenden.

Eine wiederholte Anwendung von *Regel 1* wie in Abbildung 4.17 dargestellt, ist allerdings nur möglich, wenn zuvor *Regel 2* angewendet wird. Dies bedingt sich durch die in *Regel 1* definierte negative Anwendungsbedingung, mit der ausgeschlossen wird, dass diese Regel mehrfach auf den gleichen Subgraphen angewendet wird[8]. Durch die vorangegange Anwendung von *Regel 2* sind nun aber im in Abbildung 4.17 links dargestellten Arbeitsgraphen

[8]Dies bedingt sich hier dadurch, dass die negative Anwendungsbedingung einen Knoten und eine Kante beinhalten, die auch durch die Anwendung der Regel hinzugefügt werden. Dies gilt analog für *Regel 2*.

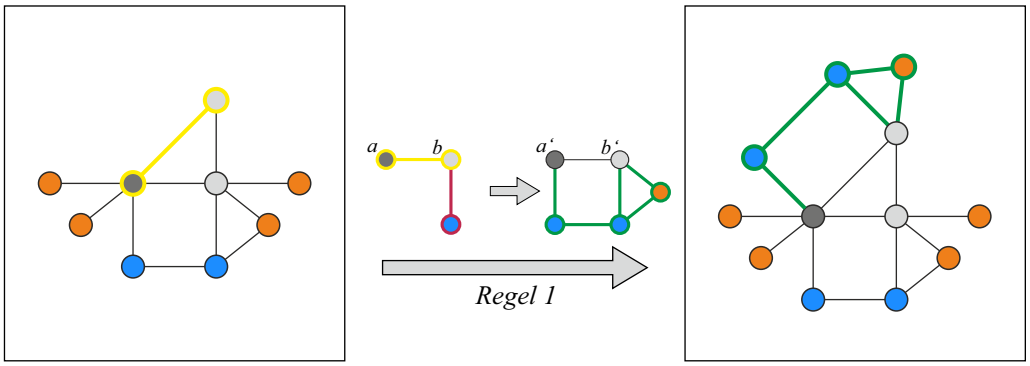


Abbildung 4.17: Durch die vorangegangene Anwendung der *Regel 2* kann *Regel 1* ein zweites Mal angewendet werden.

zwei mögliche Passungen für den Mustergraphen von *Regel 1* vorhanden, von denen nur eine durch die negative Anwendungsbedingung ausgeschlossen wird. *Regel 1* kann somit ein weiteres Mal angewendet werden. Auf den so erzeugten Graphen können nun weder *Regel 1* noch *Regel 2* angewendet werden.

Durch Graphersetzungsregeln können auch Veränderungen an einem Graphen durchgeführt werden, die über das reine Hinzufügen oder Ersetzen von Knoten und Kanten hinausgehen. Es ist beispielsweise möglich, die Werte der Attribute von Knoten und Kanten zu verändern. Hierzu muss aber neben den Graphersetzungsregeln ein sogenanntes Graph-Metamodell definiert werden, das die möglichen Typen von Knoten und Kanten in einem Graphersetzungssystem definiert.

4.2.3 Graph-Metamodell

Zur Definition der bereits beschriebenen Knoten- und Kantentypen, über die ein Graph verfügen kann, ist die Erstellung eines sogenannten Graph-Metamodells[9] nötig. Das Graph-Metamodell legt also fest, was für Graphen in einem bestimmten Kontext, also beispielsweise in einem Graphersetzungssystem betrachtet werden sollen. Wesentlich ist dabei die Definition der Typen von Knoten und Kanten, die in einem auf dem Graph-Metamodell basierenden Graphen instantiiert werden können und über welche Attribute diese Knoten und Kanten dabei verfügen. Ein Graphersetzungssystem benötigt eine solche Definition, die alle Knoten und Kanten mit einschließt, die während der Graphersetzung genutzt werden (Helms, 2013). Das Graph-Metamodell entspricht nach der OMG-Metamodell-Hierarchie[10] der M1-Ebene. Ein Graph-Metamodell kann als Klassenstruktur wie in der objektorientierten

[9] Teils wird in der einschlägigen Literatur auch nur von einem *Graphmodell* (engl. *graph model*) oder einem *Metamodell des Graphen*, gesprochen. In dieser Arbeit wird der Begriff *Graph-Metamodell* verwendet, um eine klare Abgrenzung von der M2-Ebene entsprechend der in Kapitel 2 eingeführten OMG-Metamodell-Hierarchie zu gewährleisten.

[10] Siehe auch Abschnitt 2.1.1, Seite 14.

Programmierung verstanden werden und erlaubt auch die Definition von Hierarchien zwischen den Knoten- und Kantentypen und damit eine Vererbung von Attributen.

Nach Kniemeyer (2008) kann ein Graph-Metamodell durch ein *Graph Schema* erweitert werden. Mittels dieses Schemas kann festgelegt werden, dass nur ausgewählte Typen von Kanten inzident zu Knoten eines bestimmten Typs sein dürfen. Demzufolge ist auch die Festlegung möglich, dass eine Adjazenz von Knoten nur durch Kanten eines konkret definierten Typs hergestellt werden kann. Zusätzlich kann das Schema Randbedingungen, wie z. B. die Multiplizität der Kanten[11], enthalten. Das *Graph Schema* wird im Folgenden als fester Bestandteil eines Graph-Metamodells betrachtet.

In Kapitel 6 wird das in dieser Arbeit entwickelte Graph-Metamodell ausführlich beschrieben und hinsichtlich der hier dargestellten Grundlagen eingeordnet.

4.2.4 Graphmustersuche

Der erste Schritt zur Transformation eines Arbeitsgraphen H mittels einer Graphersetzungsregel besteht in der Prüfung, ob der Mustergraph der Graphersetzungsregel in H enthalten ist. Dies wird durch eine Graphmustersuche umgesetzt und ist technisch gesehen der berechnungsaufwändigste Teil der Graphersetzung (Geiß, 2008). In Abbildung 4.18 ist links ein Graph dargestellt, dessen Teilgraphen hervorgehoben sind, die dem rechts abgebildeten Graphen entsprechen.

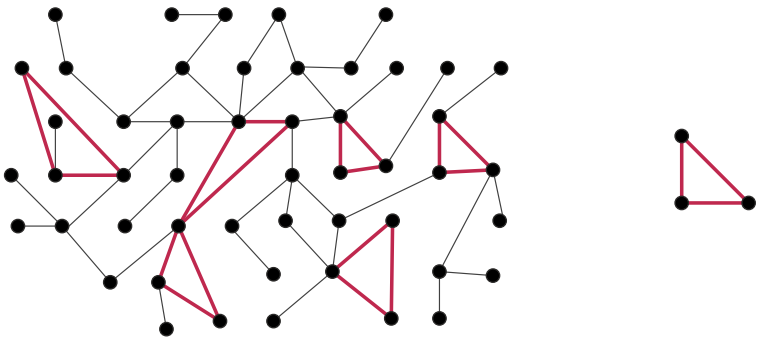


Abbildung 4.18: Ein Graph in dem seine zu einem Mustergraphen (rechts) isomorphen Teilgraphen hervorgehoben sind.

Formal betrachtet besteht die Graphmustersuche darin, festzustellen, ob zumindest ein Isomorphismus des Mustergraphen M auf einen Teilgraphen von H existiert. Dies wird als Teilgraph-Isomorphie-Problem (engl. *Subgraph isomorphism problem*) bezeichnet und ist ein NP-vollständiges Problem (Garey, 1979), welches aber für bestimmte Fälle in Polynomialzeit

[11] Die erlaubte Anzahl von Kanten eines bestimmten Typs an einem einzelnen Knoten

lösbar ist (Eppstein, 1999). Das Teilgraph-Isomorphie-Problem ist eine Generalisierung des Graph-Isomorphie-Problems (siehe 4.1.1).

Eine ausführliche Beschreibung der verschiedenen Verfahren zur Graphmustersuche, die auch das Teilgraph-Homomorphie-Problem behandelt, ist in Geiß (2008) enthalten. Eines der ersten Verfahren wird in Ullmann (1976) beschrieben und in Ullmann (2011) aktualisiert. Bei diesem Verfahren werden zur Verringerung der Laufzeit Knoten, die aufgrund der Anzahl der inzidenten Kanten nicht in Frage kommen, ausgeschlossen und anschließend iterativ Teile des Arbeitsgraphen entfernt, bis als Ergebnis nur mehr die Passungen verbleiben. Aufbauend auf den Arbeiten von Ullmann entwickelten Cordella *et al.* (2004) den VF2-Algorithmus, der die Laufzeit um einen konstanten Faktor verbessert (Geiß, 2008). Ein Verbesserung erfolgte mit dem VF3-Algorithmus (Carletti *et al.*, 2018). Durch die Verwendung von Heuristiken konnten Bonnici *et al.* (2013) den effizienten RI-Algorithmus entwickeln. Der von McCreesh *et al.* (2020) entwickelte Glasgow Subgraph Solver übertrifft nach Aussage der Autoren den RI- und den VF3 Algorithmus in vielen, aber nicht allen Fällen. Carletti *et al.* (2020) geben einen aktuellen Überblick über die performantesten Algorithmen zur Lösung des Teilgraph-Isomorphie-Problems. Die Autoren kommen zu dem Schluss, dass der RI- und der VF3-Algorithmus die anderen betrachteten Algorithmen leistungsmäßig übertreffen. Dabei benötigt der VF3-Algorithmus in der Regel am wenigsten Zeit, um eine erste Lösung zu finden.

4.2.5 Ansätze zur Graphersetzung

Nach einer erfolgreichen Graphmustersuche im Arbeitsgraphen kann die eigentliche Graphersetzungsoperation, also das Einfügen (engl. *embedding*) des Ersetzungsgraphen in den Arbeitsgraphen, durchgeführt werden. Geiß (2008) stellt fest, dass es eine Vielzahl von Ansätzen zur Graphersetzung (engl. *embedding mechanims*) gibt, die in der Literatur (beispielsweise in Rozenberg (1997)) umfassend betrachtet werden. Die Entscheidung, einen bestimmten Ansatz zu verwenden, kann allerdings kaum objektiv getroffen werden, da dies noch immer von den konkreten Anforderungen an ein Graphersetzungssystem abhängig ist. Im Vergleich zur Graphmustersuche sind die Algorithmen zur Graphersetzung komplexitätstheoretisch nicht anspruchsvoll, sondern linear zur Anzahl der betrachteten Knoten und Kanten (Geiß, 2008).

Vom gewählten Ansatz zur Graphersetzung ist abhängig, wie der Ersetzungsgraph R einer Graphersetzungsregel in den Arbeitsgraph eingefügt (oder eingebettet) wird. Man spricht daher auch vom *Einbettungsmechanismus* D, als Teil einer Graphersetzungsregel (L, D, R). Das Einfügen erfolgt, nachdem der Mustergraph L aus dem Arbeitsgraph H entfernt wurde (der verbleibende Graph wird als H^- bezeichnet). Anders ausgedrückt definiert der Ansatz zur Graphersetzung die Verknüpfung zwischen R und dem (nach Entfernung

von L) verbleibenden Arbeitsgraphen. Rozenberg (1997) unterscheidet hier zwischen zwei grundsätzlichen Verfahren: Verkleben *(Gluing)* und Neuverbinden *(Connecting)*. Diese Unterscheidung wird auch von Geiß *et al.* (2006), Kniemeyer (2008) und Wieber (2015) aufgegriffen. Es folgt eine kurze Beschreibung beider Verfahren:

- Die *Gluing Approaches* werden auch als algebraische Ansätze *(algebraic approaches)* bezeichnet (Rozenberg, 1997). Es wird dabei zwischen dem bereits erwähnten Single Pushout Approach (SPO) und dem Double Pushout Approach (DPO) unterschieden (Ehrig *et al.*, 2006; Ehrig *et al.*, 1997). Bei diesem Ansatz werden Teile von L und R direkt auf den Arbeits- bzw. den Zielgraphen abgebildet und damit deckungsgleich auf den Zielgraphen „geklebt“ (Wieber, 2015).

- *Connecting Approaches* sind Ansätze, die auch als algorithmische *(algorithmic)* oder mengentheoretische *(set theoretic)* Ansätze bezeichnet werden (Rozenberg, 1997). Man unterscheidet zwischen *Neighbourhood Controlled Embedding (NCE)* und *Hyperedge Replacement*. *NCE* kann weiterhin in *Node Label Controlled (NLC)* und *edNCE* unterschieden werden, die auch als kontextfrei bezeichnet werden. Hier wird L (genauer gesagt ein zu L isomorpher Subgraph M in H) komplett aus dem Arbeitsgraphen entfernt (wobei L z. B. im Fall von *NLC* auch nur aus einem Knoten bestehen kann) und durch R ersetzt. Dabei werden alle zu M inzidenten Kanten gelöscht, sodass genau spezifiziert werden muss, wie R durch neu zu erstellende Kanten in H^- eingefügt werden soll.

Die verschiedenen Ansätze werden in Rozenberg (1997) ausführlich und detailliert inklusive ihrer formalen Definitionen beschrieben. Im Rahmen dieser Arbeit werden nur die beiden algebraischen Ansätze weiter betrachtet. Der Grund hierfür ist, dass diese Ansätze Graphersetzungsoperationen allgemeiner (also mit weniger Einschränkungen hinsichtlich der Definition von Graphersetzungsregeln) beschreiben können. Dies zeigt sich auch darin, dass die verfügbaren Software Frameworks zu Graphersetzung (siehe Abschnitt 4.2.6) in erster Linie den SPO (und teils auch den DPO) unterstützen. SPO und DPO werden weiterhin in *Fundamentals of Algebraic Graph Transformation* (Ehrig *et al.*, 2006), das als Standardwerk der algebraischen Ansätze zur Graphersetzung betrachtet werden kann (Wieber, 2015), umfassend besprochen.

Die grundlegende Funktionsweise des SPO wurde bereits in Abschnitt 4.2.2 anhand verschiedener Beispiele beschrieben. Ein wesentliches Merkmal des SPO ist dabei der Erhaltungsmorphismus r, der einen partiellen Homomorphismus von L auf R definiert. Im Gegensatz zum SPO wird im DPO ein sogenannter Klebegraph K als Einbettungsmechanismus definiert. K kann auch als gemeinsame Schnittstelle (oder auch Schnittmenge) der beiden Seiten der Graphersetzungsregeln L und R bezeichnet werden, da von K je ein Monomorphismus[12]

[12] Bei einem Monomorphismus handelt es sich um einen injektiven Homomorphismus.

auf L sowie auf R existieren muss. Damit beschreibt K den Teil der Passung von L in H der durch die Anwendung der Graphersetzungsregeln nicht verändert wird, während $L \setminus K$ den zu löschenden Teil definiert und $R \setminus K$ den Teil, der hinzugefügt wird. Weiterhin wird eine Bedingung, die sogenannte *gluing condition* definiert, die vereinfacht besagt, dass im Arbeitsgraph H nach dem Löschen von $L \setminus K$ keine *hängenden Kanten*[13] existieren. Dies hat beispielsweise zur Folge, dass durch eine DPO Graphersetzungsregel grundsätzlich keine Knoten gelöscht werden können, die inzident zu mindestens einer Kante sind, sofern nicht mittels des Klebegraphen definiert wird, wie mit dieser Kante verfahren werden soll. Weiterhin darf es keine zwei Knoten in L geben, die auf denselben Knoten in H abgebildet werden und von denen nur einer in K enthalten ist, da dann nicht eindeutig definiert ist, ob dieser Knoten gelöscht werden soll oder nicht.

Durch dieses Verhalten wird der DPO im Vergleich zum SPO nach Wieber (2015) als „restriktiver bzw. konservativer“ beschrieben (siehe auch Ehrig *et al.* (2008) und Rozenberg (1997)), da bestimmte Szenarien ausgeschlossen werden und sie grundsätzlich umkehrbar sind (Kniemeyer, 2008). Bei Verwendung des SPO gilt die beschriebene Bedingung hingegen nicht. Stattdessen werden Knoten, bei denen nicht klar ist, ob sie gelöscht werden sollen oder nicht und hängende Kanten immer gelöscht. Durch die Verwendung von negativen Anwendungsbedingungen können aber auch bei Verwendung des SPO *gluing conditions* definiert werden. Der DPO kann damit als spezieller Fall des SPO betrachtet werden (Ehrig *et al.*, 1997).

Zur Definition der Graphersetzungsregel im Rahmen dieser Arbeit wird ausschließlich der SPO verwendet. Dies bedingt sich grundsätzlich dadurch, dass der SPO allgemeiner angewendet werden kann. Weiterhin ist die Löschung von hängenden Kanten durch den SPO insofern praktikabel, als dass im konkreten Anwendungsszenario beim Löschen eines Knotens immer auch die inzidenten Kanten gelöscht werden sollen. Bei Verwendung des DPO wäre dies nicht möglich und sogar insofern problematisch, als das ein bestimmter Knoten nicht aus einem Graphen gelöscht werden könnte, ohne dass alle potenziellen inzidenten Kanten explizit betrachtet werden müssen.

4.2.6 Software Frameworks zur Graphersetzung

Die Modellierung und Transformation von Graphen kann mit einer Vielzahl von Software-Frameworks und Werkzeugen durchgeführt werden. Die meisten der frei verfügbaren Frameworks wurden im Rahmen universitärer Projekte entwickelt. Dieser Abschnitt gibt einen Überblick über die Eigenschaften und Entwicklungszeiträume einiger der verbreitetsten Frameworks (siehe Abbildung 4.19). Aufgrund der Vielzahl weiterer Frameworks erhebt diese Auflistung keinerlei Anspruch auf Vollständigkeit, sondern legt den Fokus auf die Entwicklun-

[13] Kanten, die nur einen Endknoten haben.

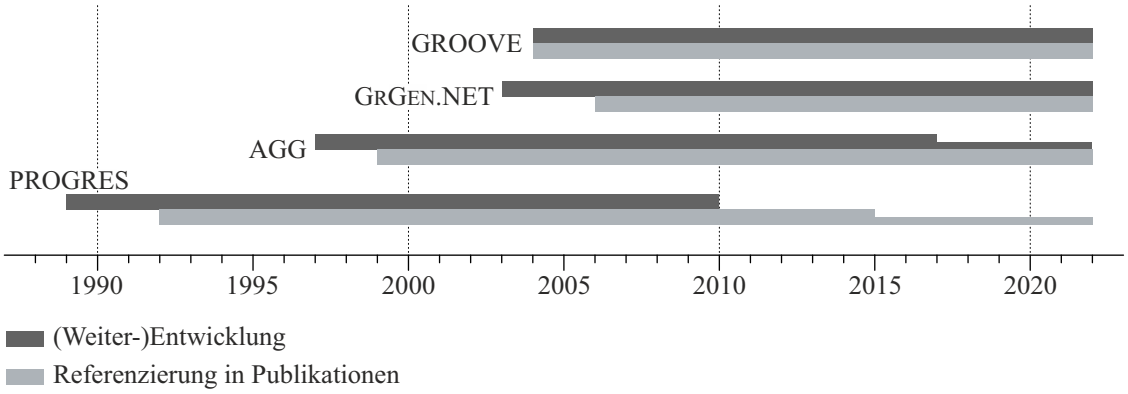


Abbildung 4.19: Nicht alle der hier vorgestellten Frameworks werden noch aktiv weiterentwickelt und gepflegt. Diese Abbildung gibt einen Überblick über die (Weiter-)Entwicklungszeiten und die Referenzierung in Forschungsarbeiten. Die kleineren Balken kennzeichnen, dass nur geringfügige Aktualisierungen veröffentlicht wurden oder dass Referenzen das Framework nur auflisten, ohne es tatsächlich zu verwenden.

gen, die vergleichsweise häufig in Forschungsprojekten eingesetzt und in wissenschaftlichen Artikeln referenziert werden. Ausführliche, aber inzwischen veraltete Auflistungen, die insbesondere ältere Frameworks beschreiben, sind in Nagl *et al.* (2000) und Rensink *et al.* (2007) enthalten. Aktuellere vergleichende Auflistungen, die neben den hier aufgeführten Frameworks auch weitere Entwicklungsarbeiten beschreiben, wurden von Aouat *et al.* (2012), Bak (2015) und Kahani *et al.* (2019) dokumentiert. Werkzeuge zur Graphtransformation werden unter anderem auch regelmäßig im jährlichen *Transformation ToolContest*[14] vorgestellt. Dieser Wettbewerb, der ab 2010 den Graph-Based Tool Contest (GraBaTs) ablöste, hat zum Ziel, die Ausdrucks- und Leistungsfähigkeit sowie die Benutzerfreundlichkeit von Transformationswerkzeugen für strukturierte Daten zu bewerten und zu vergleichen. Allen hier vorgestellten Frameworks ist gemeinsam, dass sie einen *Interpreter* bereitstellen, der lesbare Beschreibungen von Graph-Metamodellen, sowie Graphersetzungsregeln verarbeitet und dem Nutzer gegebenenfalls eine Rückmeldung zu (syntaktisch) falschen Eingaben oder Fehlern bei der Ausführung gibt. Ein *Compiler* übersetzt diese Beschreibung direkt in Quellcode oder Bibliotheken zur weiteren Verwendung. Alle hier genannten Frameworks verfügen auch über eine grafische Benutzeroberfläche zur Anzeige von Graphen und zur Visualisierung der Ausführung der Ersetzungsoperationen. Eine grafische Regeldefinition wird allerdings nur von einem Teil der Frameworks unterstützt.

Ein ausgereiftes und etabliertes Framework ist **PROGRES** (PROgrammed Graph REwriting Systems), das seit 1989 an der RWTH Aachen entwickelt wird (Zündorf, 1992). PROGRES wird allerdings nicht regelmäßig gepflegt und die aktuelle Version 9.9 wurde 2010 veröffentlicht. In aktuellen Artikeln, die einen Überblick über Frameworks zur Graphersetzung geben, wird zwar regelmäßig auf PROGRES verwiesen, Veröffentlichungen nach 2015, die die tatsächliche Verwendung in einem Projekt beschreiben, konnten allerdings nicht

[14] https://www.transformation-tool-contest.eu/aims_and_scope.html

gefunden werden. PROGRESS basiert auf gerichteten, attributierten und getypten Graphen, die umfangreiche und komplizierte Sachverhalte übersichtlich und strukturiert darstellen können. PROGRES besteht zum einen aus einer Spezifikationssprache und zum anderen aus einer komplexen, integrierten Umgebung. Das Framework erlaubt die Spezifikation eines Graph-Metamodells inklusive Vererbung und der Definition validen Kantenkardinalitäten, was beispielsweise zur Typisierung von Produktionen verwendet werden kann. Neben dem Graph-Metamodell können Ersetzungsregeln grafisch sowie textuell spezifiziert werden. Die PROGRES-Umgebung setzt sich aus drei wesentlichen Bestandteilen zusammen. Graph-Metamodell und Graphersetzungsregeln können in einem syntaxgesteuerten Editor definiert werden, der Fehler im Quelltext der PROGRESS-Syntax hervorhebt. Der Interpreter mit integrierter Graph-Visualisierung unterstützt den Benutzer bei der Fehlersuche und der Compiler übersetzt den PROGRESS-Quelltext automatisch in C- oder Java-Quellcode.

Ein weiteres verbreitetes Framework zur Graphtransformation ist der Graph Rewrite Generator **GrGen.NET** für die .NET Umgebung (Jakumeit *et al.*, 2021; Jakumeit *et al.*, 2010). GrGen.NET bietet deklarative Sprachen zur Definition von Graph-Metamodellen und Graphersetzungsregeln. Mittels der *Graph Model Language* kann ein objektorientiertes Graph-Metamodell definiert werden, das Knoten- und Kantentypen inkl. Vererbung und Attributen beschreibt. Dazu gehören auch sog. Connection Assertions, die die zulässigen Verbindungen von Knoten und Kanten in einem Graphen definieren. Das Framework bietet viele Möglichkeiten bei der Definition von Graphersetzungsregeln, einschließlich *negative application conditions*, die automatisiert mit logischen und iterativen Befehlen angewendet werden können. Die Ausführung der Graphersetzung basiert grundsätzlich auf dem SPO, es kann aber auch der DPO verwendet werden. Da GrGen.NETC#-Bibliotheken für das Graph-Metamodell und die Graphersetzungsregeln erstellt, die in den Sprachen des Frameworks definiert wurden, kann es unkompliziert in anderen Entwicklungen verwendet werden. Ein großer Vorteil von GrGen.NET ist, dass die Software einschließlich ihrer Dokumentation regelmäßig aktualisiert wird. Die zum Zeitpunkt der Drucklegung dieser Arbeit aktuellste Version 6.0 wurde im Jahr 2021 veröffentlicht.

Die Attributed Graph Grammar **AGG** ist eine regelbasierte visuelle Sprache, die einen algebraischen Ansatz zur Graphersetzung unterstützt und in Java implementiert ist (Ermel *et al.*, 1999; Runge *et al.*, 2012). Das Framework wird seit 1997 an der TU Berlin entwickelt. Das letzte größere Update für AGG stammt aus dem Jahr 2017, es wurde jedoch Anfang 2021 ein Patch veröffentlicht, was darauf schließen lässt, dass möglicherweise noch Weiterentwicklungen stattfinden. AGG erlaubt die Definition von attributierten getypten Graphen mit Vererbung. Die definierten Graphen können dabei auch Attribute in Form von Java-Objekten und -Typen enthalten. Ein Hauptmerkmal ist, dass das Framework grafische Editoren zur Erstellung von Graphen und Ersetzungsregeln sowie einen Texteditor für Java einschließlich einer Validierung und Visualisierung der Graphen bietet. AGG basiert in erster Linie auf dem Single Pushout Approach (SPO), bietet aber auch die Möglichkeit

Ersetzungen auf Basis des Double Pushout Approach (DPO) durchzuführen. Der Kern des Frameworks ist eine Engine zur Graphersetzung, die unabhängig von der grafischen Umgebung verwendet werden kann. Daher kann diese Engine auch von anderen Programmen genutzt werden.

Das **GROOVE** Tool Set (Graph-based Object-oriented Verification) wird seit 2004 entwickelt und wird weiterhin regelmäßig aktualisiert (Ghamarian *et al.*, 2012; Rensink, 2004). Aktuell ist die Version 5.8.1 verfügbar, die 2021 veröffentlicht wurde. Mit GROOVE können allerdings lediglich schlichte Graphen erstellt werden, wobei Typen und Attribute und eine entsprechende Vererbung unterstützt werden. Das Java-basierte Tool ermöglicht es, Graphen und Graphersetzungsregeln via SPO visuell zu definieren und bietet eine intuitive grafische Oberfläche, die eine grafische Bearbeitung von Regeln und Graphen ermöglicht. Ein wesentliches Merkmal besteht darin, dass GROOVE darauf ausgelegt ist, alle möglichen Resultate der Anwendung einer Menge von Ersetzungsregeln auf einen Startgraphen zu erzeugen und dies in einem übergeordneten baumartigen Graphen zu visualisieren. Damit bietet GROOVE eine intuitive Möglichkeit, die durch einen Startgraphen und Ersetzungsregeln definierte Graphgrammatik und die durch diese Grammatik beschriebene Sprache zu untersuchen.

Im Rahmen dieser Arbeit wird GrGen.NET als Framework eingesetzt, um ein Graph-Metamodell und zugehörige Ersetzungsregeln zu definieren. Ein wesentlicher Grund für diese Entscheidung besteht in der C#-basierten Implementierung von GrGen.NETund der Möglichkeit, Graph-Metamodell und Regeln als C#-Bibliotheken in einer eigenen Entwicklung einzubinden. Da auch das zur Modellerstellung genutzte CAD-System eine API auf Basis von C# bietet, war es möglich die gesamte Entwicklung innerhalb des .NET-Frameworks durchzuführen. Neben diesem Faktor bietet GrGen.NETeinen ausreichend großen Funktionsumfang, um alle im Rahmen dieser Arbeit notwendigen Definitionen von Graph-Metamodell und Ersetzungsregeln abbilden zu können. Auch die kontinuierliche Weiterentwicklung, sowie die umfangreiche und detaillierte Dokumentation waren während der Entwicklungsarbeiten von großer Hilfe.

Kapitel 5

Elemente eines parametrischen Modellierungsprozesses als Gegenstand der graphbasierten Repräsentation

In diesem Kapitel wird der Funktionsumfang eines parametrischen Modells, das durch das Graphersetzungssystem verändert werden soll, definiert. Ein wesentlicher Teil dieser Definition besteht darin, herauszuarbeiten, welche Informationen und Anforderungen eine graphbasierte Repräsentation enthalten bzw. erfüllen muss, um diesen Funktionsumfang repräsentieren zu können.

Der Funktionsumfang ergibt sich aus der Menge der Modellierungsoperationen, die durch das Graphersetzungssystem abgebildet werden. Die konkreten geometrischen Elemente, die parametrischen Zwangsbedingungen und die prozeduralen Operationen, aus denen sich ein parametrisches Modell zusammensetzt, werden dabei als Objekte bezeichnet. Mittels dieser Objekte wird die parametrische Funktionalität eines Modells sichergestellt.

Ziel dieses Kapitels ist es herauszuarbeiten und zu definieren, welche Informationen zur graphbasierten Beschreibung der einzelnen Objekte notwendig sind und welche Beziehungen zwischen den verschiedenen Objekten mithilfe des Graphen abzubilden sind. Dies stellt die Voraussetzung für die Definition des Graphersetzungssystems im nächsten Kapitel dar, da diese Informationen korrekt im Graphen abgebildet werden müssen, um ein parametrisches

Modell vollständig und eindeutig zu beschreiben. Nur wenn die hier definierten Informationen für alle Objekte vorliegen und durch den Graphen abgebildet werden, ist es möglich, den Graphen zu interpretieren, um dadurch das entsprechende parametrisch prozedurale Modell in einer CAD-Anwendung zu erstellen und weiter zu verwenden. Letztlich sind hier im Hintergrund der Modellierkern und der Geometric Constraint Solver des jeweiligen CAD-Systems (siehe auch Kapitel 3) ausschlaggebend, wobei vordergründig die von der Programmierschnittstelle (API) benötigten Informationen zur Erstellung eines Modells beachtet werden müssen.

5.1 Umfang der abzubildenden parametrischen Modelle

Prozedural und historienbasiert aufgebaute parametrische Modelle können mit aktuell verfügbaren CAD-Anwendungen eine sehr hohe Komplexität hinsichtlich der abgebildeten Formen und der parametrischen Zusammenhänge aufweisen (Marchenko *et al.*, 2011). Beispiele für solche Modelle sind sowohl im wissenschaftlichen Bereich in verschiedenen Veröffentlichungen (Lee *et al.*, 2006b; Salimzadeh *et al.*, 2020; Tang *et al.*, 2020a) als auch in der praktischen Anwendung im AEC-Sektor zu finden (Kalkan *et al.*, 2018). Diese hohe Komplexität bedingt sich im Wesentlichen durch die kontinuierliche Weiterentwicklung der CAD-Anwendungen, die es Nutzern ermöglicht, ihre Kreativität in einem computergestützten Entwurf voll umzusetzen.

Die genannte Weiterentwicklung der CAD-Anwendungen ist allerdings im Kern immer noch auf den in Kapitel 3.2 beschriebenen Grundlagen der parametrischen Modellierung aufgebaut. Beispielhaft lässt sich dies an Funktionen wie der Erstellung eines Rechtecks darstellen: Das in der CAD-Anwendung mit einer einzelnen Funktion erstellte Rechteck besteht aus Linien und geometrischen Zwangsbedingungen, welche die Anordnung dieser Linien als Rechteck sicherstellen. Weiterhin werden mithilfe sog. Features (Ranta *et al.*, 1996; Shah *et al.*, 1993) noch komplexere Operationen wie das Fasen einer Kante oder die Erstellung eines Bohrlochs für den Nutzer in einem einzelnen Schritt gebündelt: Eine gefaste Kante oder ein Bohrloch entstehen z. B. durch den Abzug eines automatisch erstellten Hilfskörpers von der bereits modellierten Geometrie.

Diese Funktionen erleichtern die bei der Modellierung notwendige Arbeit wesentlich, unterscheiden sich aber in ihrer konkreten Implementierung je nach verwendeter CAD-Anwendung – oder sind nur in einzelnen Anwendungen implementiert. Daher ist es kaum möglich, parametrische Modelle, die in einer Anwendung erstellt wurden, zu exportieren und in einer anderen Anwendung ohne Einschränkungen weiterzubearbeiten. Diese Thematik wird später in dieser Arbeit im Kontext der prototypischen Implementierung in Abschnitt 6.6 behandelt.

Im Rahmen dieser Arbeit wurde hingegen ein Ansatz entwickelt, der auf den grundlegenden Prinzipien der parametrischen Modellierung aufbaut und daher keine anwendungsspezifischen Funktionalitäten beinhaltet. Als grundlegende Prinzipien der parametrischen Modellierung werden dabei die modellierbaren geometrischen Elemente, parametrischen Zwangsbedingungen und weitere Modellierungsoperationen bezeichnet, die von den gängigen parametrischen CAD-Anwendungen unterstützt werden.

Es werden daher nur parametrische Modelle betrachtet, die sich vom hierarchischen Aufbau her aus Skizzen, Bauteilen und Baugruppen zusammensetzen. Diese können sich wie folgt aufbauen:

- Innerhalb einer Skizze können Punkte, Linien, Kreise und Kreisbögen modelliert und nur durch Zwangsbedingungen, die in der *standard geometric constraint language* (siehe Abschnitt 3.2.3) enthalten sind, angeordnet werden.
- Skizzen können innerhalb eines Bauteils auf Arbeitsebenen erstellt werden.
- Auf Basis der Skizzen wird die 3D-Geometrie eines Bauteils durch Extrusionen und Sweeps erzeugt.
- Die zur Erzeugung von Sweeps notwendigen Leitkurven können innerhalb eines Bauteils als 3D-Skizzen angelegt werden.
- Die so erstellten Bauteile können nun in einer Baugruppe kombiniert und mittels 3D-Mating-Zwangsbedingungen aneinander ausgerichtet werden.

Auch wenn dieser Umfang an Funktionalitäten und Modellierungsoperationen vollständig von gängigen parametrischen CAD-Anwendungen unterstützt wird, ist zu beachten, dass unterschiedliche CAD-Anwendungen sowohl bei der Nomenklatur als auch bei der Implementierung leichte Unterschiede aufweisen können. Die folgenden Beschreibungen dienen daher auch dazu, genau zu beschreiben, wie der Aufbau und die Funktionalitäten eines parametrischen Modells im Kontext dieser Arbeit definiert sind. Auf dieser Basis wird die graphbasierte Repräsentation im folgenden Kapitel aufgebaut. Zur Interpretation des Graphen im Kontext einer bestimmten parametrischen CAD-Anwendung, müssen daher auch immer die in diesem Kapitel definierten Gegebenheiten betrachtet werden. Bei der Erarbeitung der folgenden Inhalte wurde insbesondere darauf geachtet, dass eine graphbasierte Repräsentation erarbeitet wird, die mit verschiedenen parametrischen CAD-Anwendungen verwendbar und damit herstellerneutral ist. Dies wird auch durch die in Abschnitt 6.6 beschriebene prototypische Implementierung verdeutlicht.

5.2 Grundlegende Eigenschaften der Objekte eines parametrischen Modells

Neben den in den folgenden Abschnitten detailliert aufgeführten Eigenschaften der verschiedenen Objekttypen (geometrischen Elemente, parametrischen Zwangsbedingungen, prozeduralen Operationen) ist es notwendig, für alle Objekttypen grundlegende Eigenschaften festzuhalten:

- Jedes Objekt, das im Laufe des Modellierungsprozesses erstellt werden kann, muss über eine eindeutige Bezeichnung (ID) referenzierbar sein, damit es möglich ist, anhand dieser Bezeichnung auf das jeweilige Objekt zuzugreifen.
- Für jedes erzeugte Objekt muss der Typ dieses Objekts definiert werden können.
- Generell müssen die Beziehungen (hierarchisch, logisch, parametrisch) zwischen den Objekten abgebildet werden können.

Neben diesen grundlegenden Eigenschaften werden nun die spezifischen Eigenschaften der Objekte aufgeführt, die im Rahmen dieser Arbeit in einem Modellierungsprozess relevant sind. Ein Modellierungsprozess ist dabei die Abfolge von Modellierungsoperationen. Eine Modellierungsoperation ist ein Arbeitsschritt, der bei einer Modellierung (i. d. R. durch einen Nutzer) durchgeführt wird. Bei jedem dieser Arbeitsschritte werden Objekte erstellt oder verändert. Im Folgenden werden diese Eigenschaften dieser Objekte beschrieben, um daraus die bereits diskutierten Anforderungen an eine graphbasierte Repräsentation abzuleiten.

Die Reihenfolge, in der diese Eigenschaften beschrieben werden, orientiert sich im Wesentlichen an der hierarchischen Struktur eines parametrischen Modells. Bei der Reihung ist zusätzlich berücksichtigt, dass die Beschreibungen sinnvoll aufeinander aufbauen.

5.3 Objekte auf Baugruppenebene

Eine Baugruppe (engl. *assembly*) ist in der Hierarchie eines parametrischen Modells ein Objekt auf der höchsten Ebene. Alle weiteren Objekte sind einer bestimmten Baugruppe direkt oder indirekt zugeordnet. Die Objekte auf Baugruppenebene sind in Abbildung 5.1 dargestellt.

Dabei sind einer Baugruppe in erster Linie Bauteile (engl. *part*) direkt zugeordnet. In den gängigen parametrischen CAD-Anwendungen ist es zusätzlich möglich, eine Baugruppe in einer anderen Baugruppe zu platzieren, sodass die Baugruppen ineinander verschachtelt

sind. Im hier beschriebenen Ansatz werden allerdings nur Modelle betrachtet, die auf einer einzelnen Baugruppe basieren. Sowohl Baugruppen als auch Bauteile können zusätzlich zu ihrer ID einen Namen aufweisen. Dieser Name muss im Gegensatz zu einer ID nicht eindeutig sein. Über den Namen kann sichergestellt werden, dass eine formalisierte Modellierungsoperation nur auf bestimmte Objekte in einem Modell angewendet werden kann. Dies gilt auch für Objekte auf Bauteil- und Skizzenebene.

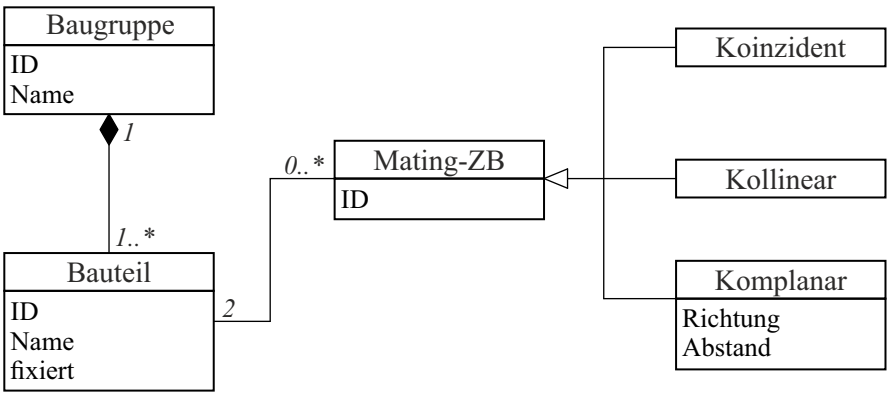


Abbildung 5.1: UML-Darstellung der Elemente auf Baugruppenebene und den Beziehungen in denen sie zueinander stehen. Eine Mating-Zwangsbedingung bezieht sich wie dargestellt immer auf genau zwei unterschiedliche Bauteile; nicht dargestellt ist, dass auf Instanzebene auch die Punkte, Kanten bzw. Flächen des Bauteils, auf die sich eine Zwangsbedingung bezieht, referenziert werden müssen.

Konkret dient eine Baugruppe im Kontext dieses Ansatzes also dazu, mehrere Bauteile zu einem Modell zusammenzufassen. Es muss somit möglich sein, einer Baugruppe ein oder mehrere Bauteile zuzuordnen. Objekte, die Teil eines Bauteils sind, werden der Baugruppe damit indirekt zugeordnet. Baugruppen müssen also als Elemente in einem Graphen, der ein Modell repräsentiert, enthalten sein.

Ein Bauteil ist ein Volumenkörper oder Volumenmodell, der auf Basis verschiedener parametrischer Modellierungsoperationen erstellt wurde. In eine Baugruppe wird das Ergebnis dieser Modellierungsoperationen als Bauteil eingefügt und im Raum platziert. Aus Sicht der Baugruppe sind die Modellierungsoperationen nicht relevant, es muss lediglich eine Zuordnung zum Bauteil vorhanden sein. Bauteile müssen also als Elemente in einem Graphen, der eine Baugruppe repräsentiert, enthalten sein.

Um die Position einer Baugruppe im Raum festzulegen, können ein oder mehrere Bauteile der Baugruppe hinsichtlich ihrer Position fixiert werden. Werden mehrere Bauteile fixiert, muss sichergestellt werden, dass keine Widersprüche, die sich durch Zwangsbedingungen ergeben können, auftreten.

Weiterhin kann die Positionierung von Bauteilen mit speziellen topologischen Zwangsbedingungen zur Verwendung in Baugruppen erfolgen. Diese Zwangsbedingungen werden als *Mating*-Zwangsbedingungen bezeichnet. Mit Hilfe dieser Zwangsbedingungen kann die

Lage der Bauteile in Abhängigkeit voneinander festgelegt werden, wodurch sich letztendlich die Geometrie einer Baugruppe ergibt. Dazu müssen Punkte, Kanten und Flächen eines Bauteils referenziert werden können, um sie zur Definition der *Mating*-Zwangsbedingungen zu verwenden.

Im Rahmen des hier beschriebenen Ansatzes werden die folgenden *Mating*-Zwangsbedingungen betrachtet (siehe Abbildung 5.2):

- Punkt zu Punkt: Ein bestimmter Punkt der Geometrie eines Bauteils muss an derselben Stelle im Raum wie ein bestimmter Punkt der Geometrie eines zweiten Bauteils liegen. Die Punkte sind damit als koinzident definiert.
  In Abbildung 5.2 sind die Punkte P_A und $P_{A'}$ koinzident. Ist der Körper A fixiert, so kann der Körper A' zwar noch um den Punkt $P_{A'}$ rotiert werden, eine Translation des Körpers in x-, y- oder z-Richtung ist jedoch nicht mehr möglich.

- Kante zu Kante: Eine bestimmte Kante der Geometrie eines Bauteils muss auf derselben Geraden wie eine bestimmte Kante der Geometrie eines zweiten Bauteils liegen. Die Kanten sind damit als kollinear definiert.
  In Abbildung 5.2 sind die Kanten L_B und $L_{B'}$ kollinear. Ist der Körper B fixiert, so kann der Körper B' nur noch um die y-Achse rotiert bzw. in y-Richtung verschoben werden. Eine Translation des Körpers in x- oder z-Richtung bzw. eine Rotation um die x- oder z-Achse ist jedoch nicht mehr möglich.
  (Die Kante L_B ist in diesem Beispiel parallel zur y-Achse.)

- Fläche zu Fläche: Eine bestimmte Fläche der Geometrie eines Bauteils muss auf derselben Ebene wie eine bestimmte Fläche der Geometrie eines zweiten Bauteils liegen. Die Flächen sind damit als komplanar definiert.
  In Abbildung 5.2 sind die Flächen F_C und $F_{C'}$ bzw. F_D und $F_{D'}$ komplanar. Ist der Körper C bzw. D fixiert, so kann der Körper C' bzw. D' nur noch in y- oder z-Richtung verschoben und um die x-Achse rotiert werden. Andere Translationen oder Rotationen sind nicht möglich.
  (Die Flächen F_C und F_D liegen in diesem Beispiel auf der yz-Ebene.)
  Weiterhin muss unterschieden werden, ob die Normalenvektoren der betroffenen Flächen in dieselbe Richtung (Fall C) oder die entgegengesetzte Richtung (Fall D) zeigen. Zusätzlich ist es auch möglich einen Abstand zwischen den komplanaren Flächen festzulegen.

Um die genannten *Mating*-Zwangsbedingungen verwenden zu können, müssen die durch die Zwangsbedingung als koinzident, kolinear bzw. koplanar definierten Punkte, Kanten bzw. Flächen der einzelnen Bauteile eindeutig referenziert werden können. Sie müssen dementsprechend als Elemente in einem Graph enthalten sein, der die übergeordneten Bauteile repräsentiert. Dabei ist es auch möglich, dass Punkte, Kanten bzw. Flächen nicht explizit

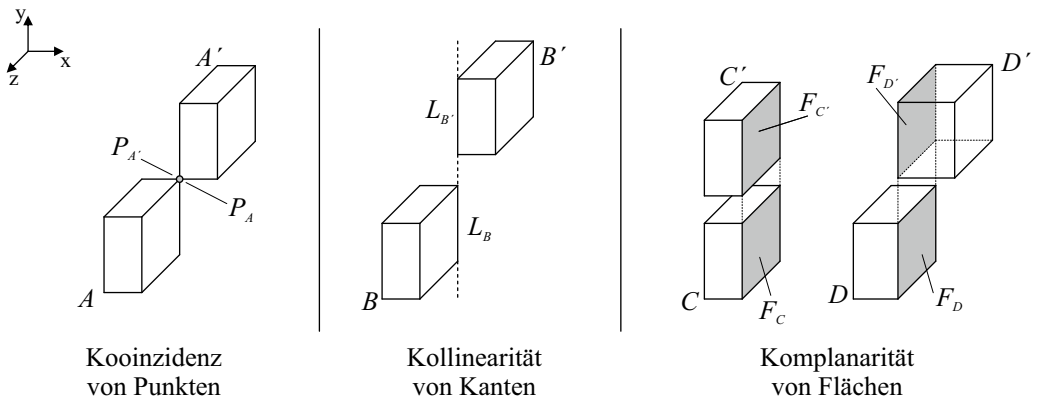


Abbildung 5.2: *Mating*-Zwangsbedingungen werden verwendet, um Körper aneinander auszurichten und hinsichtlich ihrer Freiheitsgrade einzuschränken.

enthalten sind, aber trotzdem implizit über vorhandene Elemente referenzierbar sind. Eine Fläche kann beispielsweise über die Linie, aus der sie extrudiert wurde, referenziert werden (siehe auch 6.2.6). Zusätzlich muss das Vorhandensein einer *Mating*-Zwangsbedingung im Graphen repräsentiert sein.

5.4 Objekte auf Bauteilebene

Durch ein Bauteil wird die Geometrie eines Volumenkörpers definiert. Durch die Anordnung dieser Volumenkörper in eine Baugruppe wird die finale Geometrie eines Modells zusammengesetzt. Der Aufbau der Geometrie dieser Volumenkörper und die Modellierungsoperationen durch deren Ausführung die Geometrie erzeugt wird, ist dabei jedoch hierarchisch rein innerhalb des jeweiligen Bauteils definiert. Für die Baugruppe ist lediglich das Ergebnis in Form des resultierenden Volumenkörpers relevant, da dessen Punkte, Kanten und Flächen durch die *Mating*-Zwangsbedingungen zur Positionierung genutzt werden. Die Objekte auf Bauteilebene sind in Abbildung 5.3 dargestellt.

Zur Erstellung der Geometrie eines Bauteils werden Volumenkörper auf Basis von Skizzen und verschiedenen prozeduralen Modellierungsoperationen (Sweeping Operationen, Boolesche Operationen) erstellt. Eine prozedurale Modellierungsoperation ist eine Modellierungsoperation, die nicht innerhalb einer Skizze erfolgt, sondern einen Volumenkörper entweder auf Basis einer Skizze erstellt oder diesen gegebenenfalls unter Berücksichtigung eines weiteren Volumenkörpers verändert. Sowohl die Skizzen als auch die prozeduralen Modellierungsoperationen müssen dem Bauteil zugeordnet werden können, dessen Geometrie sie erzeugen. Zusätzlich muss definiert werden welche Skizze(n) einer prozeduralen Modellierungsoperation zugrunde liegen. Weiterhin können prozedurale Modellierungsoperationen aufeinander aufbauen, sodass auch die damit einhergehenden Abhängigkeiten zwischen diesen Modellierungsoperationen definiert werden müssen.

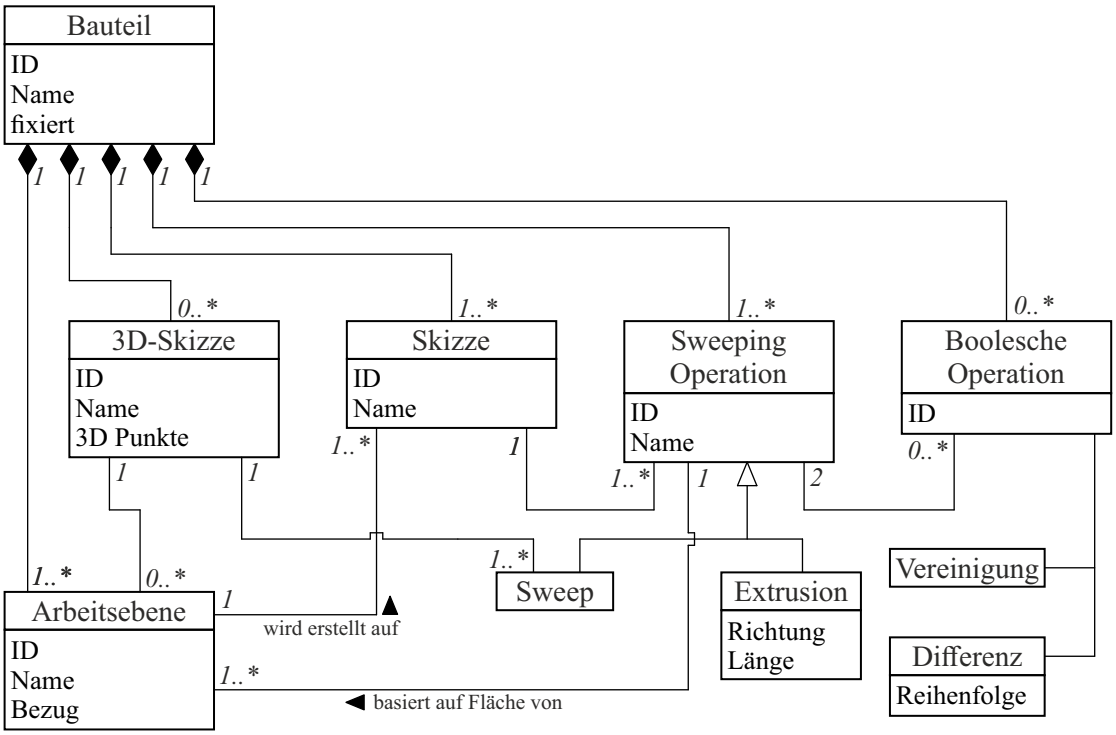


Abbildung 5.3: UML-Darstellung der Elemente auf Bauteilebene und der Beziehungen, in denen sie zueinander stehen.

Skizzen dienen bei der Erstellung eines Volumenkörpers entweder als Grundfläche oder als Pfad, entlang dem aus der Grundfläche ein dreidimensionaler Körper erzeugt wird, indem die Grundfläche entlang des Pfades in den Raum gezogen wird. Wie genau die Geometrie dabei erstellt wird, bedingt sich durch die verwendete Modellierungsoperation. Wird eine Skizze als Grundfläche für eine Modellierungsoperation verwendet, handelt es sich um eine 2D-Skizze. Weiterhin können Skizzen unabhängig von Ebenen auf Basis von Linien und Kurven im Raum angelegt werden. Diese Skizzen werden hier als 3D-Skizzen bezeichnet[1] und über Punkte im Raum definiert. 3D-Skizzen können als Pfade für Sweeps genutzt werden. Die einzelnen Punkte können dabei entweder durch lineare Segmente miteinander verbunden werden oder durch einen Spline interpoliert werden. 2D-Skizzen werden im Rahmen dieser Arbeit auch nur als Skizzen bezeichnet, 3D-Skizzen werden zum besseren Verständnis explizit so benannt.

Jede 2D- oder 3D-Skizze muss dem Bauteil, in dem sie sich befindet, zugeordnet werden können. Bei 2D-Skizzen ist es zusätzlich notwendig, eine Ebene als sogenannte Arbeitsebene (engl. *workplane*) zu definieren, auf der die Skizze angelegt wird. Auch die Arbeitsebene muss wiederum dem Bauteil, in dem sie sich befindet, zugeordnet werden können. Weiterhin muss die Beziehung zwischen Skizze und Arbeitsebene definiert werden können. Es ist dabei

[1] Diese Bezeichnung wird beispielsweise auch in Autodesk Inventor und Solidworks von Dassault Systèmes verwendet.

auch möglich, dass mehrere Skizzen auf derselben Arbeitsebene angelegt werden. Über die Arbeitsebene, auf der eine Skizze erstellt wird, definiert sich auch das zweidimensionale Koordinatensystem der Skizze.

Im System jedes Bauteils bilden drei Arbeitsebenen in der xy-, der xz- bzw. der yz-Ebene die Ausgangsbasis für die Erstellung von Skizzen. Diese Arbeitsebenen bestimmen sich durch die x-, y- und z-Achse, die das lokale Koordinatensystem des Bauteils definieren. Das Koordinatensystem der auf diesen Arbeitsebenen erstellten Skizzen leitet sich dabei aus den Achsen, durch die die Arbeitsebene definiert ist, ab.

Weiterhin kann eine Arbeitsebene durch die Fläche eines bereits modellierten Volumenkörpers definiert werden. Die Abhängigkeit der Arbeitsebene von der entsprechenden Fläche muss dabei festgelegt werden. Das Koordinatensystem der Skizze muss dabei in Abhängigkeit der Begrenzungen (Kanten) der Fläche des Volumenkörpers definiert werden können. Nur so kann bei der Erstellung der Skizzenelemente sichergestellt werden, dass diese korrekt positioniert werden. Alternativ können geometrische Elemente in einer Skizze rein durch Projektion (projizierte Geometrie) erstellt werden, sodass es nicht notwendig ist, ein Koordinatensystem zu definieren. Eine weitere Möglichkeit besteht darin, eine Arbeitsebene auf Basis einer 3D-Skizze zu erstellen. Die Arbeitsebene ist dabei von dem durch die 3D-Skizze definierten Pfad selbst und von einem Punkt auf diesem Pfad abhängig und wird an der Stelle des angegeben Punkts lotrecht zum Pfad definiert.

Im Rahmen dieses Ansatzes werden die prozeduralen Modellierungsoperationen Extrusion, Sweep und Kombination (Boolesche Operationen) berücksichtigt. Diese werden im Folgenden als *Sweeping Operationen* bezeichnet. Die grundlegende Funktionsweise dieser Modellierungsoperationen im Kontext der parametrischen Modellierung ist in Kapitel 3 beschrieben.

Um eine Extrusion zu erstellen, muss einerseits die Skizze, die dieser Extrusion zugrunde liegt, angegeben werden und weiterhin die Länge, um die extrudiert wird, sowie die Extrusionsrichtung bekannt sein. Die Richtung der Extrusion mit einer positiven Länge definiert sich dabei durch den Normalenvektor der Skizze (der sich wiederum durch das Koordinatensystem der Skizze definiert). Um in die entgegengesetzte Richtung zu extrudieren, muss ein negativer Wert für die Extrusionslänge angegeben werden.

Auch zur Erstellung eines Sweeps muss die Skizze angegeben werden, die die Grundfläche des Sweeps definiert. Die Leitkurve, entlang der der Sweep verläuft, wird über eine 3D-Skizze bestimmt. Die Skizze, die als Grundfläche dient muss dabei auf einer Arbeitsebene erstellt werden, welche durch dieselbe 3D-Skizze und einen Punkt, der auf dieser 3D-Skizze liegt, definiert wurde. Beginn und Ende des Sweeps bestimmen sich durch die Länge der durch die 3D-Skizze definierten Leitkurve.

Sind innerhalb eines Bauteils mehrere Volumenkörper modelliert worden, können diese durch Boolesche Operationen (Vereinigung oder Differenz; Schnittmengen werden nicht betrachtet) miteinander kombiniert werden. Dabei muss definiert werden, welche Boolesche Operation für diese Modellierungsoperation verwendet werden soll und welche Volumenkörper kombiniert werden sollen. Im Fall einer Differenz muss die Reihenfolge berücksichtigt werden, damit eindeutig ist, welcher Körper abgezogen wird.

Die Volumenkörper, die als Ergebnis von Modellierungsoperationen entstehen, müssen innerhalb eines Bauteils eindeutig bestimmbar sein, damit sie von weiteren Modellierungsoperationen als deren Grundlage referenziert werden können. Zusätzlich ist es notwendig, dass Punkte, Kanten und Flächen der Geometrie der Volumenkörper eindeutig referenzierbar sind. Nur dann ist es beispielsweise möglich, eine Arbeitsebene auf Basis einer bestimmten Fläche eines bestimmten Volumenkörpers zu erstellen. Auch ist die Referenzierung von Punkten oder Kanten zur Erstellung von projizierter Geometrie in weiteren Skizzen notwendig, welche im Abschnitt 5.5.2 beschrieben wird. Diese Referenzierbarkeit muss durch die graphbasierte Repräsentation eines Volumenkörpers innerhalb eines Bauteils ermöglicht werden.

5.5 Objekte auf 2D-Skizzenebene

Eine (parametrische) 2D-Skizze ist Bestandteil eines Bauteils und bildet die Grundlage für Modellierungsoperationen zur Erstellung eines Volumenkörpers. Eine Skizze muss sowohl dem Bauteil, zu dem sie gehört, als auch der Arbeitsebene, auf der sie erstellt wurde, zugeordnet werden können. Die Zugehörigkeit zu einem Bauteil ist dabei implizit durch die Zugehörigkeit der Arbeitsebene zum übergeordneten Bauteil gegeben.

Zur Erstellung einer Skizze werden geometrische Elemente und parametrische Zwangsbedingungen verwendet. Die Menge der geometrischen Elemente einer Skizze setzt sich aus dem für einen Betrachter sichtbaren Teil dieser Skizze zusammen. Im Rahmen dieser Arbeit werden Punkte, Linien, Kreise und Kreisbögen betrachtet. Auf Basis dieser geometrischen Grundelemente kann einerseits bereits eine Vielzahl von Modellen erzeugt werden. Weiterhin sind diese Grundelemente in den gängigen parametrischen CAD-Anwendungen implementiert.

Die Topologie der geometrischen Elemente wird durch die parametrischen Zwangsbedingungen definiert. Erst mit Hilfe dieser Zwangsbedingungen werden die parametrischen Eigenschaften der Skizze definiert und die Skizze kann durch die Variation der dabei angelegten Parameter dimensionaler Zwangsbedingungen kontrolliert verändert werden.

Die geometrischen Elemente müssen der Skizze, zu der sie gehören, zugeordnet werden können. Auch wenn die finale Positionierung der Elemente letztlich durch parametrische

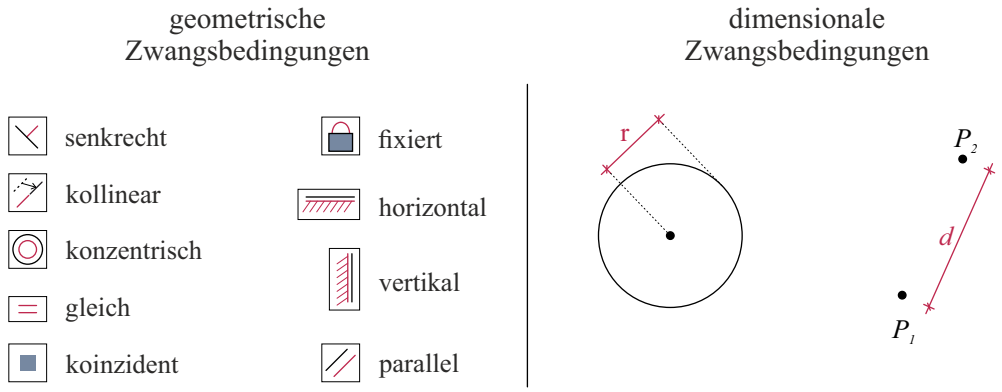


Abbildung 5.4: Geometrische und dimensionale Zwangsbedingungen inklusive der Symbole bzw. Bemaßungen durch die sie üblicherweise dargestellt werden. Die Symbole orientieren sich an der Darstellung in gängigen CAD-Systemen.

Zwangsbedingungen definiert wird, ist es notwendig, die Elemente zum Zeitpunkt ihrer Erstellung über Koordinaten zu positionieren. Das entsprechende Vorgehen und die daraus ableitbaren Anforderungen an eine graphbasierte Repräsentation werden in Abschnitt 5.5.1 beschrieben.

Wie in Abschnitt 3.2.3 eingeführt, gliedern sich Parametrische Zwangsbedingungen in geometrische und dimensionale Zwangsbedingungen (siehe Abbildung 5.4). Dimensionale Zwangsbedingungen werden eingesetzt, um Abstände zwischen geometrischen Elementen oder die Dimensionen dieser Elemente zu kontrollieren. Zusätzlich zu den dimensionalen Zwangsbedingungen können geometrische Zwangsbedingungen definiert werden. Sie dienen dazu, die eigentliche Anordnung der geometrischen Elemente, aus denen die Skizze aufgebaut ist, zu definieren. Beispielsweise können damit zwei Linien als parallel oder orthogonal definiert werden.

In den folgenden Abschnitten werden die geometrischen Elemente und die parametrischen Zwangsbedingungen detailliert beschrieben und dabei herausgearbeitet, welche Informationen in einer graphbasierten Repräsentation enthalten sein müssen. Eine Übersicht über alle Objekte auf Skizzenebene ist in Abbildung 5.5 dargestellt.

5.5.1 Geometrische Elemente

In diesem Abschnitt werden die geometrischen Elemente, aus denen sich eine Skizze zusammensetzen kann, beschrieben. Relevant sind dabei für jedes Element einerseits die Koordinaten, durch die die initiale Position des Elements im 2D-Koordinatensystem der Skizze beschrieben wird - also die Position, an der das Element bei seiner Erstellung platziert wird. Diese Position kann sich jedoch durch die Erstellung parametrischer Zwangsbedingungen verändern. Dieses Verhalten wird im folgenden Abschnitt detailliert beschrieben. In den nachfolgenden Abschnitten sind die geometrischen Elemente inklusive der benötigten

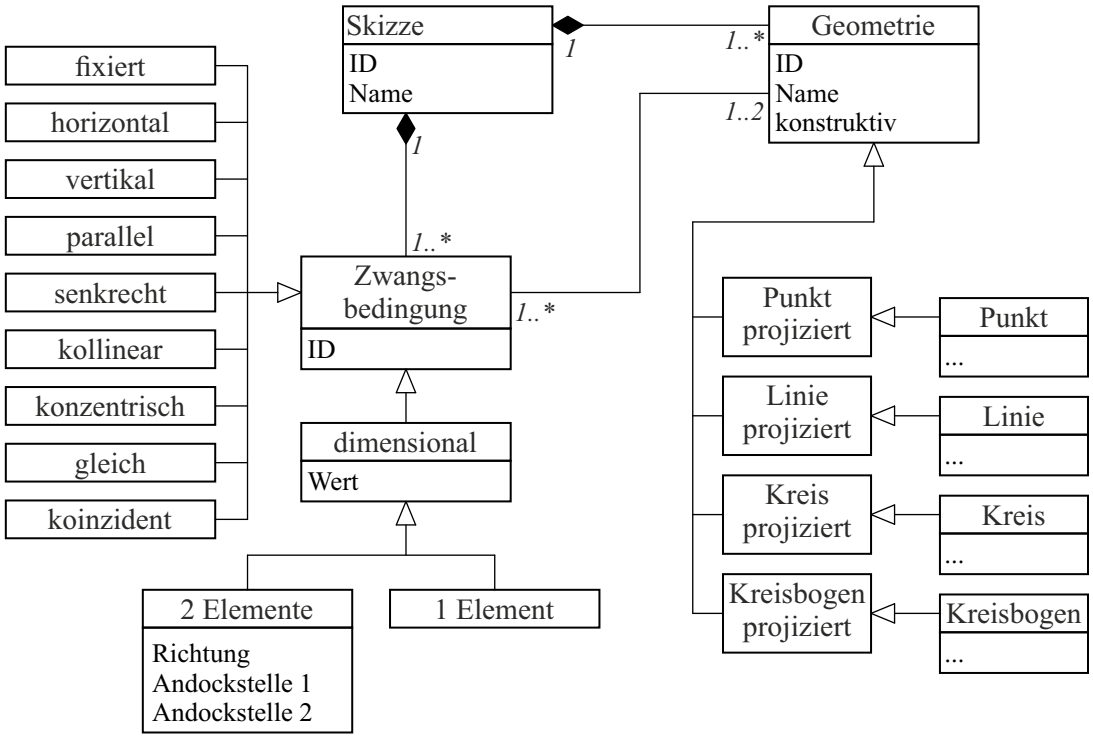


Abbildung 5.5: UML-Darstellung der Elemente auf Skizzenebene und der Beziehungen in denen sie zueinander stehen. Die notwendigen Informationen zu den Andockstellen und der initialen Position der geometrischen Elemente ist hier nicht detailliert aufgeführt.

Koordinaten aufgelistet. Dabei wird auch für jedes Element beschrieben, wie es mittels parametrischer Zwangsbedingungen in die Topologie der Skizze eingebunden werden kann. Alle Elemente können als *konstruktiv* und damit als Konstruktionsgeometrie deklariert werden. Ein so deklariertes Element wird in einer Skizze als Konstruktionshilfe verwendet, aber bei einer Extrusion oder bei einem Sweep nicht berücksichtigt.

Initiale Positionierung der geometrischen Elemente

Im Kontext der parametrischen Modellierung zeichnen sich Skizzen entsprechend der Definition im vorangegangenen Kapitel 3 unter anderem durch ihre flexible Veränderbarkeit aus, die durch die Modifikation von Parametern gesteuert werden kann. Die Koordinaten eines geometrischen Elements entsprechen daher nicht zwangsläufig der Position, an der ein Element initial während der Modellierung platziert wurde. Bei einer vollständig parametrisierten Skizze bestimmt sich die Position eines Elementes vielmehr allein durch die Koordinaten die der GCS für dieses Element als Teil der Lösung des Constraint-Problems berechnet. Die initialen Koordinaten werden also auf Basis dieser Berechnungen überschrieben.

Da bei der Lösung des Constraint-Problems allerdings auch die initiale Positionierung der geometrischen Elemente berücksichtigt wird, ist es notwendig, diese Informationen durch die graphbasierte Repräsentation abzudecken.

Insofern ergibt sich für jedes geometrische Element die Anforderung, dass die initiale Position definierbar sein muss. Dabei ist zu beachten, dass es sich bei den Koordinaten, durch die sich diese initiale Position bestimmt, um temporäre Koordinaten handelt. Wird die Position des Elements durch parametrische Zwangsbedingungen, die in nachgelagerten Modellierungsoperationen definiert werden, verändert, so ändern sich auch die Koordinaten des Elements. Die temporären Koordinaten der initialen Position des Elements sind daher nur bei der Erstellung des geometrischen Elements relevant. Dieses Verhalten muss sowohl bei der Konzeption der graphbasierten Repräsentation als auch bei der Interpretation des Graphen berücksichtigt werden.

Andockstellen

Weiterhin wird bei der Beschreibung der geometrischen Elemente jeweils angegeben, wie ein Element in die Topologie der jeweiligen Skizze eingebunden werden kann. Dazu wurde zwischen den verschiedenen „Teilen“ eines Elements unterschieden, auf die sich eine parametrische Zwangsbedingung beziehen kann. Die Teile der geometrischen Elemente werden im Folgenden als *Andockstellen* bezeichnet. Diese Andockstellen sind beim Anlegen parametrischer Zwangsbedingungen von immenser Wichtigkeit, da sich je nach Kombination von geometrischer Zwangsbedingung, geometrischem Element und Andockstelle unterschiedliche parametrische Zusammenhänge und damit unterschiedliche Topologien der Skizze ergeben.

Auch bei der Beschreibung der parametrischen Zwangsbedingungen in Abschnitt 5.5.2 wird daher beschrieben, welches parametrische Verhalten eine Zwangsbedingung im Hinblick auf unterschiedliche Andockstellen verschiedener geometrischer Elemente hervorruft. Damit dieses Verhalten korrekt aus einer graphbasierten Repräsentation abgeleitet werden kann, muss im Graphen definiert werden können, auf welche Andockstelle eines geometrischen Elements sich eine parametrische Zwangsbedingung bezieht.

Punkte

Das einfachste geometrische Element, das zum Aufbau einer Skizze genutzt werden kann, ist der Punkt im zweidimensionalen Raum. Die initiale Position des Punkts in der Skizze wird durch eine x- und eine y-Koordinate bestimmt (Abbildung 5.6). Diese Koordinaten müssen also im Graphen repräsentiert werden können. Parametrische Zwangsbedingungen, durch die die Position des Punkts verändert werden können beziehen sich immer auf den Punkt selbst. Als Andockstelle kommt also implizit nur der Punkt selbst in Frage. So kann

ein Punkt beispielsweise als koinzident zu einem anderen Punkt definiert werden oder der Abstand zwischen zwei Punkten festgelegt werden.

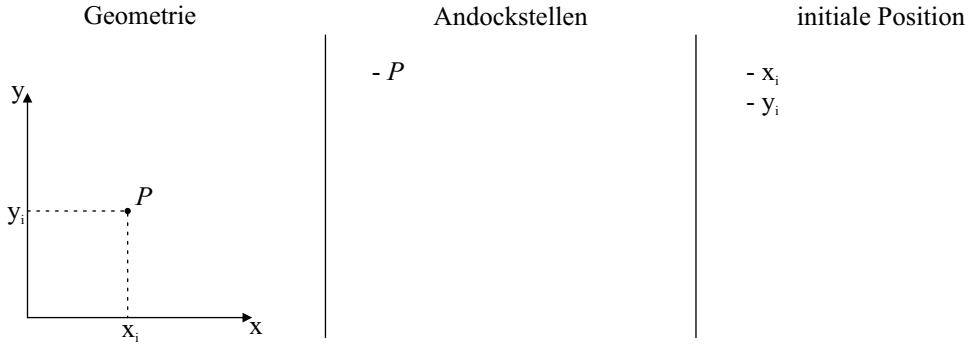


Abbildung 5.6: Geometrie, Andockstelle, und die zur initialen Positionierung notwendigen Informationen eines Punkts.

Ein Punkt muss der Skizze, zu der er gehört, eindeutig zugeordnet werden können. Die Zuordnung zu Bauteil und Baugruppe wird dadurch implizit definiert. Dies gilt analog für die folgenden geometrische Elemente.

Linien

Eine Linie ist die gerade Verbindung zweier Punkte. Formal gesehen handelt es sich hierbei um einen Geradenabschnitt, der durch die beiden Punkte definiert wird. Die initiale Positionierung der Linie in einer Skizze bestimmt sich durch die x- und y-Koordinaten der beiden Punkte. Diese vier Koordinaten müssen also im Graphen repräsentiert werden können. Rein geometrisch ist es nicht notwendig, zwischen den beiden Punkten zu unterscheiden, da die Linie in jedem Fall eine gerade Strecke ist, die durch zwei Endpunkte begrenzt wird. Ein Vertauschen der Endpunkte führt nicht zu einer Veränderung der Geometrie dieser Strecke.

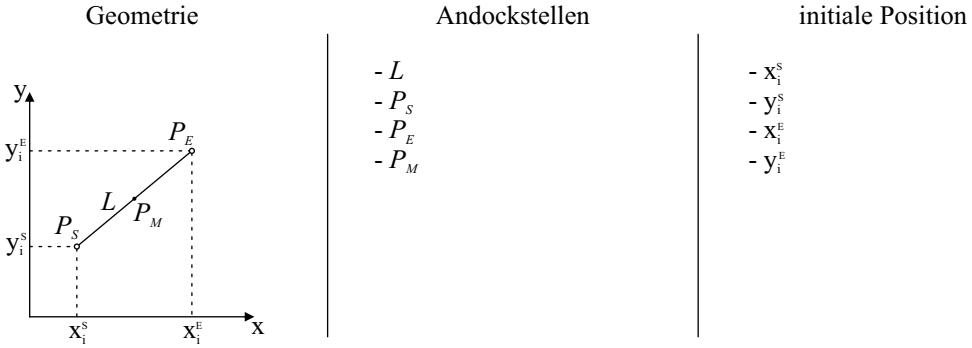


Abbildung 5.7: Geometrie, Andockstellen, und die zur initialen Positionierung notwendigen Informationen einer Linie.

Um eine eindeutige Referenzierung der beiden Punkte in einer graphbasierten Repräsentation zu ermöglichen, werden die beiden Punkte aber im Folgenden als Start- und Endpunkt der

Linie bezeichnet (Abbildung 5.7). Dies ist auch notwendig, um eine Linie über parametrische Zwangsbedingungen in die Topologie einer Skizze einzubinden. Die Zwangsbedingungen können sich dabei entweder auf den Start-, den End- oder den Mittelpunkt einer Linie beziehen. Die Koordinaten des Mittelpunkts müssen allerdings nicht explizit im Graphen repräsentiert werden, da sie implizit bekannt sind. Zusätzlich können sich bestimmte Zwangsbedingungen auch auf die Linie als Ganzes beziehen. Dies ist beispielsweise dann der Fall, wenn eine Linie als horizontal definiert wird oder ein Punkt an einer beliebigen Stelle auf der Linie liegen muss. So können die Position und damit auch die Länge einer Linie über parametrische Zwangsbedingungen festgelegt und gesteuert werden. Die Länge der Linie kann über eine dimensionale Zwangsbedingungen gesteuert werden, die sich nur auf die Linie bezieht.

Kreise

Formal gesehen beschreibt ein Kreis die Menge aller Punkte auf einer Ebene, die denselben Abstand zu einem bestimmten Punkt auf dieser Ebene haben. Dieser Punkt ist der Mittelpunkt des Kreises, der Abstand der Radius. Zur initialen Positionierung des Kreises sind die x- und y-Koordinaten des Mittelpunkts sowie der Betrag des Radius notwendig (Abbildung 5.8).

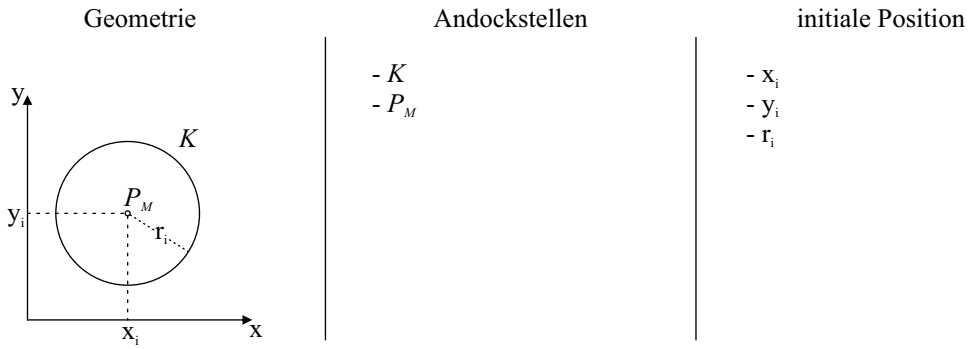


Abbildung 5.8: Geometrie, Andockstellen, und die zur initialen Positionierung notwendigen Informationen eines Kreises.

Parametrische Zwangsbedingungen können sich entweder auf den Mittelpunkt des Kreises oder auf den Kreis selbst beziehen. So kann einerseits die Position des Kreises (durch Referenzierung des Mittelpunkts) durch parametrische Zwangsbedingungen festgelegt oder verändert werden. Weiterhin kann auch der Kreis selbst (die Kreislinie K) referenziert werden, um beispielsweise einen Punkt als koinzident zur Kreislinie zu definieren. Der Radius des Kreises kann über eine dimensionale Zwangsbedingungen gesteuert werden, die sich nur auf den Kreis bezieht.

Kreisbögen

Ein Kreisbogen ist eine zusammenhängende Teilmenge des Kreises zwischen zwei Punkten, die auf der Kreislinie liegen. Um einen Kreisbogen zu definieren, müssen daher ein Mittelpunkt und die beiden Punkte definiert werden. Der Radius des Kreisbogens bestimmt sich dabei implizit. Für eine eindeutige Definition muss allerdings zwischen einem Start- und einem Endpunkt des Kreisbogens unterschieden werden. In einer graphbasierten Repräsentation des Kreisbogens müssen also die initialen x- und y-Koordinaten von Mittel-, Start- und Endpunkt enthalten sein. Es wird weiterhin festgelegt, dass der Kreisbogen den Start- und den Endpunkt im Uhrzeigersinn verbindet. Neben den drei genannten Punkten können sich parametrische Zwangsbedingungen auch auf den Kreisbogen K beziehen. Der Radius des Kreisbogens kann über eine dimensionale Zwangsbedingungen gesteuert werden, die sich nur auf den Kreisbogen bezieht (Abbildung 5.9).

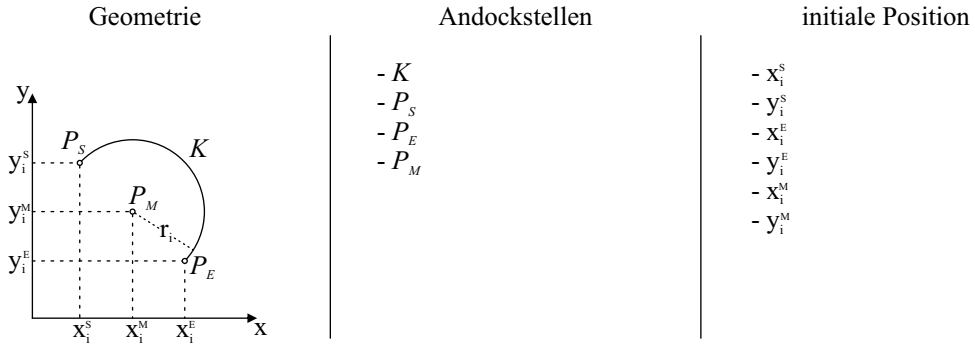


Abbildung 5.9: Geometrie, Andockstellen, und die zur initialen Positionierung notwendigen Informationen eines Kreisbogens.

Generell ist die Konstruktion eines Kreisbogens im Zuge einer manuellen Modellierung auch über andere Eingaben möglich. Beispielsweise können drei Punkte, durch die der Bogen verlaufen muss, angegeben werden oder der Mittelpunkt, der Radius und der Mittelpunktswinkel definiert werden. Da sich die finale Position des Kreisbogens aber letztlich durch die Verwendung von Zwangsbedingungen ergibt, werden die zusätzlichen Konstruktionsmethoden bei der graphbasierten Repräsentation hier nicht berücksichtigt.

Einordnung

Auch wenn die Menge der hier betrachteten geometrischen Grundelemente nur Punkte, Linien, Kreise und Kreisbögen umfasst, kann auf Basis dieser Elemente bereits eine sehr große Anzahl an Anwendungsfällen der Modellierung abgedeckt und damit die Basis für eine Vielzahl von Volumenmodellen erstellt werden. Betrachtet man verschiedene parametrische CAD-Anwendungen und die dort modellierbaren Elemente, lässt sich weiterhin feststellen, dass die hier genannten geometrischen Elemente in allen Anwendungen verfügbar sind.

Zusätzlich sind sie in allen Anwendungen so definiert, dass durch die korrekte Interpretation der graphbasierten Repräsentation die gleiche Skizze entsteht. Dies bedingt sich auch dadurch, dass die Menge der auf dem Markt etablierten parametrischen Modellierkerne ohnehin recht überschaubar ist[2]

Hier nicht betrachtete geometrische Elemente sind im Wesentlichen Ellipsen und verschiedene Freiformkurven. Eine Erweiterung des Konzepts um diese Elemente ist allerdings generell möglich.

5.5.2 Parametrische Zwangsbedingungen

Erst durch die Verwendung von parametrischen Zwangsbedingungen wird eine statische Zeichnung zu einer Skizze im Sinne der parametrischen Modellierung. Im Folgenden wird beschrieben, welche Zwangsbedingungen im Rahmen dieser Arbeit berücksichtigt werden und welches parametrische Verhalten durch die Verwendung dieser Zwangsbedingungen bei der Erstellung von Modellen erzielt werden kann.

Dimensionale Zwangsbedingungen

Ein wesentliches Merkmal parametrischer Modelle besteht darin, dass es möglich ist, die Abstände zwischen geometrischen Elementen oder die Dimensionen eines einzelnen Elements zu kontrollieren. Diese Abstände werden hierfür über einen Parameter mit einem Wert versehen, durch den sich die Größe des jeweiligen Abstandes dynamisch bestimmen und damit interaktiv steuern lässt. Zur Definition solcher parameterabhängigen Dimensionen werden dimensionale Zwangsbedingungen eingesetzt. Während das fundamentale Prinzip dieser Zwangsbedingungen in Kapitel 3 beschrieben wurde, wird im Folgenden dargestellt, wie und in welchem Umfang dimensionale Zwangsbedingungen im Rahmen dieser Arbeit verwendet werden.

Generell wird dabei zwischen zwei wesentlichen Szenarien bei der Erstellung unterschieden. Einerseits kann sich eine dimensionale Zwangsbedingung auf nur ein geometrisches Element beziehen, um eine Abmessung dieses Elements über einen Parameter steuerbar zu machen. So kann die Länge einer Linie oder der Radius eines Kreises bzw. eines Kreisbogens definiert werden. In diesem Fall ist implizit über die Art des Elements ableitbar, auf welche Dimension sich die Zwangsbedingung bezieht.

Wird eine dimensionale Zwangsbedingung allerdings genutzt, um einen Abstand zwischen zwei geometrischen Objekten zu definieren, muss zwischen verschiedenen Fällen unterschieden werden. Hier werden auch die bereits beschriebenen Andockstellen relevant, da diese

[2]Hauptsächlich werden ACIS, Parasolid, C3D, Shape Manager eingesetzt (Ushakov, 2012).

festlegen, auf welchen Teil eines geometrischen Elements sich die dimensionale Zwangsbedingung bezieht. Eine dimensionale Zwangsbedingung bezieht sich damit immer auf einen Abstand, der sich durch zwei Kombinationen aus geometrischem Element und Andockstelle bestimmt. Die möglichen Kombinationen sind in der folgenden Grafik (Abbildung 5.10) schematisch dargestellt. Einige dieser Kombinationen werden allerdings in der praktischen Anwendung kaum (und im Kontext der in dieser Arbeit betrachteten Beispiele nie) eingesetzt. Dies bedingt sich einerseits dadurch, dass der festzulegende Abstand bei bestimmten Kombinationen meist mehrdeutig ist, und andererseits durch die Möglichkeit, dasselbe Ergebnis durch andere Zwangsbedingungen herzustellen. Diese Fälle werden daher nicht betrachtet und sind in Abbildung 5.10 mit „o“ gekennzeichnet.

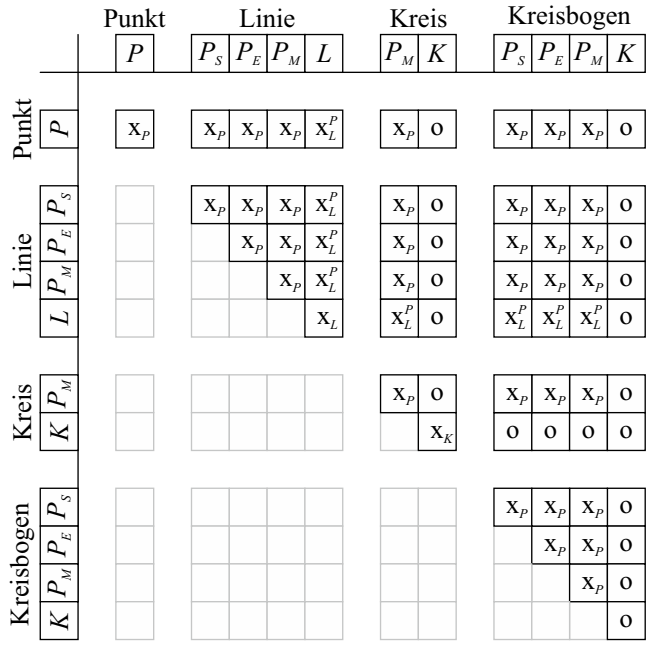


Abbildung 5.10: Mögliche Verwendung dimensionaler Zwangsbedingung zur Beschreibung des Abstands zweier geometrische Elemente unter Berücksichtigung der Andockstellen.
Der ausgegraute Bereich ist symmetrisch zum befüllten Bereich.

Bei den mit x_P gekennzeichneten Kombinationen handelt es sich um Abstände zwischen zwei Punkten, wobei diese Punkte Teil eines geometrischen Elements sein können und über die Andockstellen des Elements definiert werden. Eine dimensionale Zwangsbedingung definiert in diesem Fall entweder den minimalen, den horizontalen oder den vertikalen Abstand zwischen den beiden Punkten. Für die Zwangsbedingung muss festgelegt werden, welchen dieser Abstände sie beschreiben soll.

Die mit x_L^P gekennzeichneten Kombinationen beziehen sich auf Abstände zwischen einem Punkt und einer Linie, wobei der Punkt Teil eines geometrischen Elements sein kann. Eine so angelegte dimensionale Zwangsbedingung definiert den Abstand zwischen dem Punkt und einer Geraden, auf der die Linie liegt. Wird ein Lot durch den Punkt auf die Gerade

gefällt, so erhält man einen weiteren Punkt S – der durch die Zwangsbedingung definierte Abstand ist $d(S, P)$.

Um einen Abstand zwischen zwei Linien (x_L) zu definieren (ohne, dass sich dieser Abstand auf einen konkreten Punkt auf diesen Linien bezieht), müssen diese Linien parallel sein. Durch die Zwangsbedingung wird der Abstand zwischen den beiden Geraden, auf denen die Linien jeweils liegen, beschrieben.

Auch der Abstand zwischen den Kreislinien zweier Kreise (x_K) kann festgelegt werden. Die Zwangsbedingung beschreibt dabei den kleinstmöglichen Abstand zweier Punkte, die auf den Kreislinien liegen. Dieser ist durch den Abstand der Schnittpunkte der Kreislinien mit einer Geraden, die durch die beiden Kreismittelpunkte verläuft, definiert. Im Kontext dieser Arbeit werden dabei nur Kreise betrachtet, deren Kreislinien sich nicht schneiden. Liegt ein Kreis vollständig innerhalb der Kreislinie des zweiten Kreises, werden nur Fälle von konzentrischen Kreisen betrachtet.

Zur Repräsentation einer dimensionalen Zwangsbedingung in einem Graphen muss also definiert sein, auf welches geometrische Element bzw. auf welche geometrischen Elemente sich diese bezieht. Bei bestimmten Elementen muss außerdem festgelegt sein, welche Andockstellen genutzt werden.

Zusätzlich muss der Parameter, durch den die Größe des Abstands beschrieben werden kann, repräsentiert werden. Bei diesem Parameter kann es sich um einen numerischen Wert handeln, der die Größe des Abstands, der durch die Zwangsbedingung beschrieben wird, bestimmt. Es ist auch möglich, den Parameter durch eine Formel zu definieren, die sich auf die Größe einer anderen parametrischen Zwangsbedingung bezieht. Der Abstand wird dadurch von einer oder mehreren anderen Abständen in einer Skizze abhängig. So kann beispielsweise festgelegt werden, dass die Länge eines Rechtecks immer doppelt so groß ist wie die Breite. Als Resultat wird die betroffene Skizze so verändert, dass der beschriebene Abstand die korrekte Größe aufweist. Bei der Beschreibung eines Parameters durch eine Formel ist anzumerken, dass die so entstandene Abhängigkeit nur in eine Richtung (unidirektional) wirken kann. Die Abmessung eines geometrischen Elements A, dessen Abmessung von der eines anderen Elements B abhängig ist, kann nur durch die Veränderung von B geändert werden.

Es ist weiterhin möglich, den Parameter einer Zwangsbedingung bei ihrer Erstellung nicht durch einen Wert oder eine Formel zu definieren. In diesem Fall nimmt der Parameter als Wert automatisch die Größe des Abstands zum Zeitpunkt der Erstellung an.

Geometrische Zwangsbedingungen

Während die dimensionalen Zwangsbedingungen die Abstände der geometrischen Elemente innerhalb einer Skizze definieren, werden die geometrischen Zwangsbedingungen genutzt, um die Ausrichtung der geometrischen Elemente zueinander festzulegen. Im Kontext dieser Arbeit werden die folgenden geometrischen Zwangsbedingungen betrachtet[3]:
fixiert, horizontal, vertikal, parallel, senkrecht, kollinear, koinzident, konzentrisch, gleich

Es gilt dabei für alle geometrischen Zwangsbedingungen, die sich auf zwei geometrische Elemente beziehen, dass sie *bidirektional* wirken. Das heißt, dass sich eine Änderung an einem der verknüpften geometrischen Elemente immer auch auf das andere Element auswirkt. Wird beispielsweise eine Linie L_1 rotiert, die als parallel zu einer anderen Linie L_2 definiert ist, so wird eine entsprechende Rotation auf die Linie L_2 angewandt.

Im Folgenden werden die in dieser Arbeit verwendeten geometrischen Zwangsbedingungen hinsichtlich des durch sie hervorgerufenen parametrischen Verhaltens beschrieben. Dabei werden auch die Anforderungen an eine graphbasierte Repräsentation herausgearbeitet.

fixiert *(engl. locked)*
Durch die Anwendung der Zwangsbedingung *fixiert* wird ein geometrisches Element in einer Skizze an seiner aktuellen Position festgehalten. Die Zwangsbedingung kann sich dabei nur auf ein geometrisches Element beziehen. Je nach Andockstelle des fixierten geometrischen Elements bestimmt sich, wie genau das Element fixiert wird.

Bezieht sich die Zwangsbedingung auf einen Punkt bzw. eine Andockstelle in Form eines Punktes, so werden die Koordinaten dieses Punktes fixiert. Wird bei einer Linie die gesamte Linie als Andockstelle gewählt, so wird die Linie hinsichtlich der Geraden, auf der sie liegt, fixiert, die Position des Start- und des Endpunkts der Linie kann allerdings noch verändert (also auf dieser Gerade verschoben) werden. Wird die Kreislinie eines Kreises als Andockstelle gewählt, so ist dieser Kreis sowohl hinsichtlich seines Radius als auch hinsichtlich der Koordinaten des Kreismittelpunkts fixiert. Das gleiche gilt für einen Kreisbogen, wobei hier die Position des Start- und des Endpunkts noch insofern variiert werden können, als dass sich die Länge des Kreisbogens verändern lässt.

In einer graphbasierten Repräsentation muss eindeutig definiert werden können, auf welchen der genannten Fälle sich eine fixiert-Zwangsbedingung bezieht.

horizontal
Wird eine Linie mit der Zwangsbedingung *horizontal* versehen, muss diese Linie parallel zur x-Achse des Koordinatensystems der Skizze sein. Bezieht sich die Zwangsbedingung

[3] Es wurden diese Zwangsbedingungen ausgewählt, da sie in der von Bhatt *et al.* (2013) definierten *standard geometric constraint language* enthalten sind und damit in allen branchenüblichen parametrischen CAD-Systemen verwendet werden können.

auf zwei Punkte oder zwei Andockstellen in Form eines Punktes, so sind diese Punkte horizontal entsprechend dem Koordinatensystem der Skizze ausgerichtet (sie haben also die gleich y-Koordinate).

vertikal
Wird eine Linie mit der Zwangsbedingung *vertikal* versehen, muss diese Linie parallel zur y-Achse des Koordinatensystems der Skizze sein. Bezieht sich die Zwangsbedingung auf zwei Punkte oder zwei Andockstellen in Form eines Punktes so sind diese Punkte vertikal entsprechend dem Koordinatensystem der Skizze ausgerichtet (sie haben also die gleich x-Koordinate).

parallel
Die Zwangsbedingung *parallel* bezieht sich immer auf zwei Linien. Für diese Linien wird durch diese Zwangsbedingung definiert, dass sie parallel zueinander sind. In einer graphbasierten Repräsentation muss abgebildet werden, auf welche Linien sich die Zwangsbedingung bezieht. Die Andockstellen spielen hier keine Rolle.

senkrecht *(engl. perpendicular)*
Die Zwangsbedingung *senkrecht* bezieht sich immer auf zwei Linien. Für diese Linien wird durch diese Zwangsbedingung definiert, dass sie senkrecht aufeinander stehen. In einer graphbasierten Repräsentation muss abgebildet, werden auf welche Linien sich die Zwangsbedingung bezieht. Die Andockstellen spielen hier keine Rolle.

kollinear
Die Zwangsbedingung *kollinear* bezieht sich immer auf zwei Linien. Für diese Linien wird durch diese Zwangsbedingung definiert, dass sie auf derselben Gerade liegen. In einer graphbasierten Repräsentation muss abgebildet, werden auf welche Linien sich die Zwangsbedingung bezieht. Die Andockstellen spielen hier keine Rolle.

konzentrisch
Die Zwangsbedingung *konzentrisch* bezieht sich immer auf zwei geometrische Elemente, die entweder Kreise oder Kreisbögen sein müssen. Für diese Kreise bzw. Kreisbögen wird durch diese Zwangsbedingung definiert, dass sie den selben Mittelpunkt haben. In einer graphbasierten Repräsentation muss abgebildet, werden auf welche geometrischen Elemente sich die Zwangsbedingung bezieht. Die Andockstellen spielen hier keine Rolle.

gleich *(engl. equal)*
Die Zwangsbedingung *gleich* bezieht sich immer auf zwei geometrische Elemente, die entweder Linien oder Kreise sein müssen. Für diese beiden Linien bzw. Kreise wird durch diese Zwangsbedingung definiert, dass sie auch nach der Veränderung eines der beiden Elemente die gleiche Länge bzw. den gleichen Radius haben. Hier ist insbesondere hervorzuheben, dass diese beiden Längen *bidirektional* voneinander abhängen. Es ist somit gleichgültig, bei welchem der beiden betroffenen geometrischen Elemente sich der Abstand ändert, da diese

Veränderung jeweils auf die Länge bzw. den Radius des anderen Elements propagiert wird. Dies unterscheidet diese Zwangsbedingung von den dimensionalen Zwangsbedingungen, bei denen sich Veränderungen immer nur in eine Richtung auswirken können.

koinzident

Die Zwangsbedingung *koinzident* bezieht sich immer auf zwei geometrische Elemente. Durch die Anwendung der Zwangsbedingung kann einerseits definiert werden, dass bestimmte Punkte der geometrischen Elemente zusammenfallen müssen, das heißt dass sie an derselben Stelle liegen und demnach die gleichen Koordinaten haben. Diese Zwangsbedingung wird daher auch als mit dem Begriff *zusammenfallend* bezeichnet. Welche Punkte der geometrischen Elemente koinzident sind, kann mithilfe der Andockstellen festgelegt werden. Weiterhin kann die Zwangsbedingung verwendet werden, um festzulegen, dass ein Punkt auf einer Linie, auf der Kreislinie eines Kreises oder auf der Kreislinie eines Kreisbogens liegen muss. Auch dieser Fall kann über die Angabe der entsprechenden Andockstelle festgelegt werden. Die verschiedenen Möglichkeiten der Anwendung dieser Zwangsbedingung sind in der folgenden Abbildung 5.11 abgebildet und anschließend nochmals genau definiert.

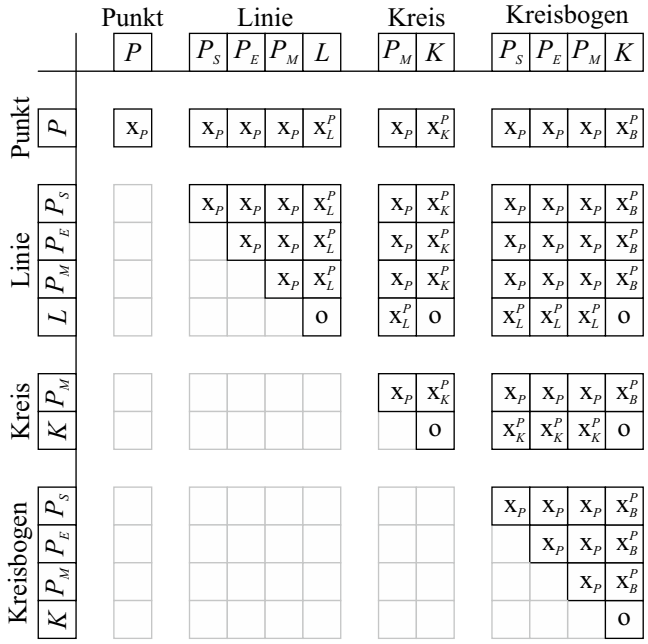


Abbildung 5.11: Mögliche Verwendung der Zwangsbedingung *koinzident* zur Definition zusammenfallender Punkte geometrischer Elemente unter Berücksichtigung der Andockstellen.
Der ausgegraute Bereich ist symmetrisch zum befüllten Bereich.

Bei den mit x_P gekennzeichneten Kombinationen bezieht sich die Zwangsbedingung auf zwei Punkte, wobei diese Punkte Teil eines geometrischen Elements sein können und über die Andockstellen des Elements definiert werden. Es kann damit beispielsweise definiert werden, dass ein Punkt und der Mittelpunkt eines Kreises oder der Startpunkt einer Linie koinzident sein müssen. Es ist aber genauso möglich festzulegen, dass der Startpunkt einer Linie und der Endpunkt eines Kreisbogens dieselbe Position haben müssen.

Im Fall der mit x_L^P gekennzeichneten Kombinationen bezieht sich die Zwangsbedingung auf einen Punkt, der Teil eines geometrischen Elements sein kann und eine Linie. So wird definiert, dass sich der Punkt an einer beliebigen Stelle auf der Linie befinden muss. Beispielsweise kann so festgelegt werden, dass sich der Startpunkt einer Linie an einem beliebigen Punkt auf einer anderen Linie befinden muss. Dabei kann sich die Position des Punktes aber nachträglich durch Veränderungen der Skizze entlang der Linie verschieben.

Analog wird bei der mit x_K^P gekennzeichneten Kombinationen der Fall betrachtet, dass sich die Zwangsbedingung auf einen Punkt, der Teil eines geometrischen Elements sein kann, und auf die Kreislinie eines Kreises bezieht. Damit wird festgelegt, dass sich der Punkt an einer beliebigen Stelle auf der Kreislinie befinden muss, also das z. B. der Startpunkt einer Linie auf der Kreislinie eines Kreises liegen muss. Durch die Veränderungen der Skizze kann sich die Position des Punktes auch nach Anwendung der Zwangsbedingung entlang der Kreislinie verschieben.

Die mit x_B^P gekennzeichneten Kombinationen entsprechen im Wesentlichen der im vorigen Absatz beschriebenen Kombination, wobei in diesem Fall festgelegt wird, dass der Punkt in diesem Fall auf dem Kreisbogen liegen muss.

Welche dieser Kombinationen durch die Zwangsbedingung *koinzident* beschrieben wird, ergibt sich immer aus der Kombination der geometrischen Elemente und der Andockstellen. Somit muss eine graphbasierte Repräsentation der Zwangsbedingung einerseits den Bezug zu den beiden betroffenen geometrischen Elementen und andererseits die Andockstellen, die verwendet werden sollen, abbilden können.

Projizierte Geometrie

Die Verwendung der beschriebenen parametrischen Zwangsbedingungen innerhalb einer Skizze bietet bereits eine Vielzahl von Möglichkeiten bei der Erstellung parametrischer Modelle. Es ist allerdings nicht möglich, parametrische Zusammenhänge zwischen den geometrischen Elementen verschiedener Skizzen zu definieren. Damit kann eine Änderung, die sich durch die Variation eines Parameters ergibt, nicht automatisiert auf geometrische Elemente, die in einer weiteren Skizze erstellt wurden, propagiert werden. Dies führt zu Einschränkungen bei verschiedenen Szenarien der Modellierung.

Zur Definition von parametrischen Zwangsbedingungen zwischen geometrischen Elementen, die in unterschiedlichen Skizzen erstellt werden, kann daher sogenannte projizierte Geometrie verwendet werden. Das bedeutet, dass ein bestehendes geometrisches Element in einer Skizze in eine zweite Skizze projiziert wird. In dieser zweiten Skizze wird ein entsprechendes geometrisches Element erstellt. Dieses neu erstelle Element ist vom ursprünglichen Element

abhängig, sodass eine Änderung der Position oder Größe dieses Elements automatisch zu einer entsprechenden Änderung des abhängigen Elements führt.

Beispielhaft ist eine solche Projektion in Abbildung 5.12 dargestellt. Hier wird eine Linie von der ursprünglichen Skizze in eine weitere Skizze projiziert, wobei die beiden Skizzen auf parallelen Arbeitsebenen liegen. Generell ist dieses Vorgehen auch für nichtparallele Skizzen – was zur Verzerrung der geometrischen Elemente führen kann – anwendbar, dieser Fall wird jedoch in der vorliegenden Arbeit nicht betrachtet.

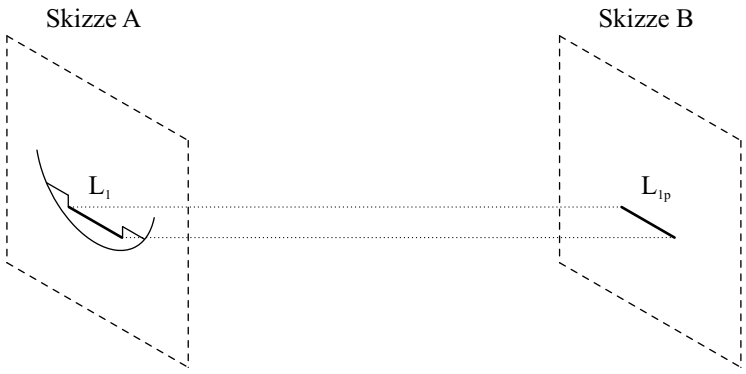


Abbildung 5.12: Projektion einer Linie aus Skizze A in Skizze B.

Um projizierte geometrische Elemente in einem Graphen zu repräsentieren, muss die Beziehung des projizierten Elements zum ursprünglichen geometrischen Element definiert werden können. Es muss also eine Beziehung zwischen geometrischen Elementen unterschiedlicher Skizzen repräsentiert werden können. Weiterhin muss das projizierte geometrische Element der Skizze, in der er es erstellt wird, zugeordnet werden können.

Projizierte geometrische Elemente verfügen über die gleichen Andockstellen wie die entsprechenden Ursprungselemente. Sie können über Zwangsbedingungen mit anderen geometrischen Elementen verknüpft werden, sodass sich eine Änderung des Ursprungselemente auf die gesamte Geometrie der Skizze in der das projizierte Elemente liegt, auswirkt. Hier ist jedoch zu beachten, dass Änderungen dabei nur in eine Richtung propagiert werden können, das heißt dass es nicht möglich ist, durch die Veränderung des projizierten Elements das Ursprungselement zu verändern. Es handelt sich also in diesem Fall um eine unidirektionale Abhängigkeit.

5.6 Integration von Semantik

Prinzipiell wird bereits durch die eindeutige Bezeichnung, die wie anfangs beschrieben zur Identifikation aller Objekte in einem parametrischen Modell notwendig ist, eine grundlegende Semantik zusätzlich zur reinen Geometrie des Modells definiert. Zusätzlich kann jedem Objekt eine beliebige Anzahl von Eigenschaften zugeordnet werden. Dadurch lassen sich

letztlich die praktischen Einsatzmöglichkeiten des Modells deutlich erweitern, da Eigenschaften wie beispielsweise Materialkennwerte an die Bestandteile eines Modells angehängt werden können. Als eine weitere Anforderung an die graphbasierte Repräsentation wird damit festgelegt, dass es möglich sein muss, Bauteil- und Baugruppen-Objekten verschiedene Eigenschaften zuweisen zu können. Jede Eigenschaft sollte dabei über eine Bezeichnung und einen Wert beschrieben werden können, wobei der Wert sowohl ein Text als auch eine Zahl sein kann.

5.7 Zusammenfassung

Insgesamt ergeben sich auf Basis der in diesem Kapitel beschriebenen Objekte und ihrer Zusammenhänge eine Vielzahl von Anforderungen, die durch das Graphersetzungssystem abgebildet werden müssen.

Während die Repräsentation der Objekte an sich weitgehend direkt aus den hier beschriebenen Anforderungen abgeleitet werden kann, muss bei den verschiedenen Anforderungen an die Beschreibung der Zusammenhänge und Beziehungen der Objekte zueinander sehr genau auf die Eindeutigkeit der Repräsentation geachtet werden. Gerade bei der Beschreibung der parametrischen Zwangsbedingungen muss exakt definiert werden, wie die Zwangsbedingungen in welchen Kombinationen mit geometrischen Objekten, deren Andockstellen und anderen Zwangsbedingungen welches parametrische Verhalten hervorrufen, da in einem parametrischen System auch kleine Abweichungen, unterschiedliche Resultate hervorrufen können.

Letztlich müssen diese Eventualitäten sowohl bei der Definition der graphbasierte Repräsentation berücksichtigt werden und zusätzlich auch bei der anschließenden Erstellung von Graphersetzungsregeln Beachtung finden. Nur so kann sichergestellt werden, dass bei Verwendung des Graphersetzungssystem nur Graphen erzeugt werden können, aus denen eindeutig ein valides parametrisches Modell erzeugt werden kann, dass auch in einer parametrische CAD-Anwendung als evaluiertes Modell erstellbar ist.

Kapitel 6

Ein Graphersetzungssystem zur Automatisierung parametrischer Modellierungsoperationen

Ein Graphersetzungssystem ermöglicht es, einen Graphen durch die Anwendung von klar definierten Regeln zu transformieren. Eine entscheidende Rolle spielt dabei der Typ des Graphen, der durch das Graphersetzungssystem erstellt und verändert werden soll.

Um die im vorangegangenen Kapitel definierten Anforderungen an eine graphenbasierte Repräsentation zu erfüllen, ist es notwendig, einen getypten und attributierten Graphen zu verwenden. Dazu muss festgelegt werden, aus welchen Typen von Knoten und Kanten sich dieser Graph zusammensetzen kann und welche Attribute die Knoten und Kanten besitzen können. Erst anschließend können konkrete Graphersetzungsregeln definiert werden, da diese die Zusammensetzung des Graphen reflektieren müssen, um angewendet werden zu können.

Formal betrachtet ist ein Graphersetzungssystem – wie bereits in Kapitel 4 beschrieben – eine Menge M von Graphersetzungsregeln $p\colon L \to R$. Jede Graphersetzungsregel p besteht aus dem Mustergraphen L und dem Ersetzungsgraphen R. Durch die Anwendung von p auf einen Graphen G wird dieser Graph durch das Hinzufügen oder Entfernen von Knoten oder Kanten verändert. L muss dazu aus solchen Knoten und Kanten zusammengesetzt sein, die auch in G enthalten sind, um angewendet werden zu können. Daher muss für ein Graphersetzungssystem insgesamt die Menge an möglichen Knoten- und Kantentypen definiert werden und sowohl Muster- als auch Ersetzungsgraph können nur aus Knoten und

Kanten dieser Typen bestehen. Die Definition der Knoten- und Kantentypen erfolgt im Graph-Metamodell des Graphersetzungssystems und muss demnach finalisiert sein, bevor Graphersetzungsregeln formuliert werden können.

Die Typen von Knoten und Kanten leiten sich aus den im vorangegangenen Kapitel erarbeiteten Voraussetzungen ab. So muss für jedes beschriebene Objekt und jede Beziehung zwischen den Objekten ein Knoten- oder Kantentyp festgelegt werden. Für diese Knoten- und Kantentypen wird dann weiterhin definiert, wie durch sie das jeweilige Objekt in einem Modell innerhalb des Graphen – auch im Kontext zu anderen Objekten und deren Repräsentation – eindeutig repräsentiert werden kann.

In diesem Kapitel wird das im Rahmen dieser Arbeit entwickelte Graphersetzungssystem formal definiert und anhand von konkreten Beispielen zur Verdeutlichung beschrieben. Der formalen Definition vorangestellt sind dabei zwei Abschnitte, in denen übergeordnet auf den Entwicklungsprozess und die Konzeption des Graphen eingegangen wird. Auf dieser Basis wird ausgehend von den Festlegungen im vorangegangenen Kapitel das Graph-Metamodell des Graphersetzungssystems aufgebaut und im Kontext der formalen Definition eingeordnet. Anschließend werden Graphersetzungsregeln zur Beschreibung von verschiedenen Modellierungsoperationen konzipiert, die einen Anwendungsfall des Graphersetzungssystems verdeutlichen. Abschließend wird auf die technische Umsetzung des Graphersetzungssystems auch im Zusammenhang mit den parametrischen CAD-Anwendungen, in denen auf Basis der graphbasierten Repräsentation parametrische Modelle erstellt werden können, eingegangen.

Es muss weiterhin einleitend darauf hingewiesen werden, dass der Begriff *Modell* in diesem Kapitel in unterschiedlicher Hinsicht verwendet wird. Einerseits bezeichnet er die implizite Beschreibung eines geometrischen Modells, das mithilfe parametrischer und prozeduraler Modellierungsoperationen erstellt wurde und hier konsequent als parametrisches Modell bezeichnet wird. Es handelt sich dabei um die abstrahierte Darstellung eines realen oder gedachten Gegenstandes, wobei in der Regel entweder ein Bauwerk oder ein Teil eines Bauwerks gemeint ist[1]. Andererseits wird dieses parametrische Modell in einer zweiten Stufe der Abstraktion durch einen Graphen repräsentiert, der aber letztlich nur eine andere Form zur Abbildung des realen Objekts ist. Damit ist auch der Graph ein Modell (nämlich *das Modell eines parametrischen Modells*), wobei dieses Modell als graphbasierte Repräsentation bezeichnet wird, um eine Verwechslung zu vermeiden[2]. Der Vollständigkeit halber ist schon an dieser Stelle anzumerken, dass zusätzlich der in Kapitel 4 eingeführte Begriff Graph-Metamodell verwendet wird, bei dem es sich entsprechend seiner Definition auch um ein Modell handelt[3]. Eine Verwechslung mit den beiden erstgenannten Modellen sollte aber durch den Kontext der Verwendung ausgeschlossen sein. Zur Verdeutlichung der

[1] M0-Ebene entsprechend der OMG-Metamodell-Hierarchie, siehe Abschnitt 2.1.1, Seite 14.

[2] Entsprechend der OMG-Metamodell-Hierarchie handelt es sich bei der graphbasierten Repräsentation weiterhin um die M0-Ebene.

[3] Dies entspricht nach der OMG-Metamodell-Hierarchie der M1-Ebene. Hier ist zu beachten, dass sich die Verwendung des Begriffs *Metamodell* entsprechend der OMG-Metamodell-Hierarchie hinsichtlich der

verschiedenen Verwendungen des Modellbegriffs, der damit einhergehenden Abstraktion und der Einordnung nach der OMG-Metamodell-Hierarchie ist in Abbildung 6.1 eine Übersicht gegeben.

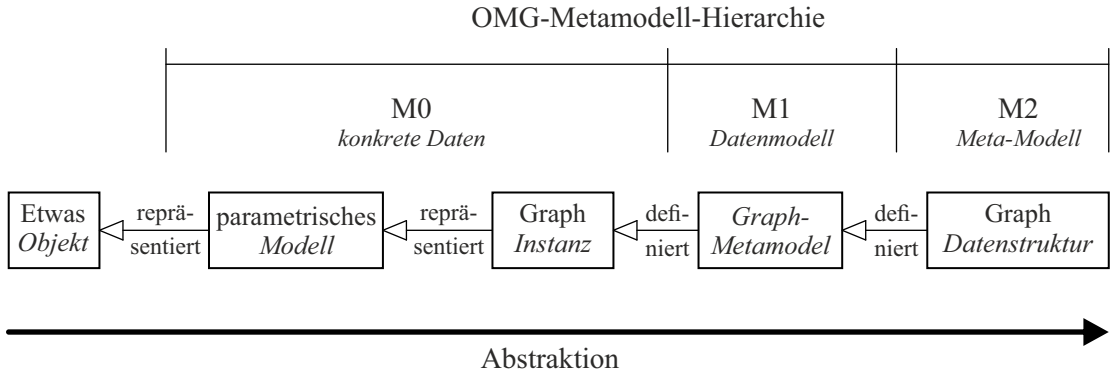


Abbildung 6.1: Modell-Hierarchie im Kontext dieser Arbeit: Ein *Objekt* wird durch ein parametrisches CAD-*Modell* repräsentiert. Dieses Modell wird wiederum durch eine bestimmte *Instanz* eines Graphen repräsentiert, dessen generelle Beschaffenheit durch ein Graph-*Metamodell* definiert wird. Durch die Definition dieses Graph-Metamodells wird die allgemeine *Datenstruktur* eines Graphen konkretisiert.

6.1 Vorgehensweise und Entwicklungsprozess

Ein großer Vorteil der Verwendung von Graphen und Graphersetzungssystemen zur Modellierung komplexer Systeme besteht darin, dass Graphen solche Systeme einerseits formal repräsentieren können, und andererseits in der Möglichkeit, das System durch den Graphen zu visualisieren. Im Gegensatz zu anderen Formen der formalen Repräsentation ermöglicht die Visualisierung eines Graphen ein vergleichsweise schnelles Begreifen des durch den Graphen beschrieben Zustands eines Systems (Purchase, 2000; Tamassia, 2000), wenn der Graph beispielsweise mittels eines geeigneten Algorithmus gezeichnet wird[4] (Battista *et al.*, 1994; Blythe *et al.*, 1996). Beispielhaft können hier visuelle Programmiersprachen (Schiffer, 1998) oder die Unified Modeling Language (UML) genannt werden. Der Zustand eines Systems ist in beiden Beispielen erst durch die Visualisierung einfach ersichtlich. Zusätzlich können gewünschte Veränderungen direkt an der Visualisierung des Graphen über eine graphische Benutzeroberfläche vorgenommen werden.

Über ein Graphersetzungssystem können weiterhin auch Operationen, die den Zustand eines Systems verändern, formal repräsentiert und zusätzlich auch visuell aufbereitet werden. Mit Hilfe eines Graphersetzungssystems kann somit der Zustand eines Systems vor einer

Abstraktionsebene vom Begriff *Graph-Metamodell* im Kontext von Graphersetzungssystemen unterscheidet, siehe auch Abbildung 6.1.

[4]Man spricht hier von der Lesbarkeit eines Graphen, die sich durch verschiedene Ästhetiken der Graphzeichnung ausdrücken lässt (Battista *et al.*, 1994).

möglichen Veränderung, der Zustand nach dieser Veränderung und die Veränderung selbst sowohl visuell als auch formal beschrieben werden.

6.1.1 Konzept

Der Zustand eines Systems ist im hier betrachteten Fall immer ein parametrisches Modell, das durch die Verwendung parametrischer und prozeduraler Modellierungstechniken erstellt wird. Damit verschiedene Zustände dieses Systems durch Graphen repräsentiert werden können, müssen solche Graphen also ein parametrisches Modell abbilden. Aufgrund der Beschaffenheit von Modellen, die mit parametrischen und prozeduralen Modellierungstechniken erstellt wurden, kann insbesondere auch die Abfolge der Modellierungsschritte, die zur Erstellung des Modells ausgeführt wurden, als Historie des Modells in einer Repräsentation mit betrachtet werden. Es kann daher nicht nur das implizite Endergebnis einer Abfolge von Modellierungsoperationen durch den Graphen repräsentiert werden, sondern die gesamte Entstehungshistorie des Modells – soweit dies mit den im vorangegangenen Kapitel beschriebenen Objekten und den Methoden der parametrischen Modellierung möglich ist.

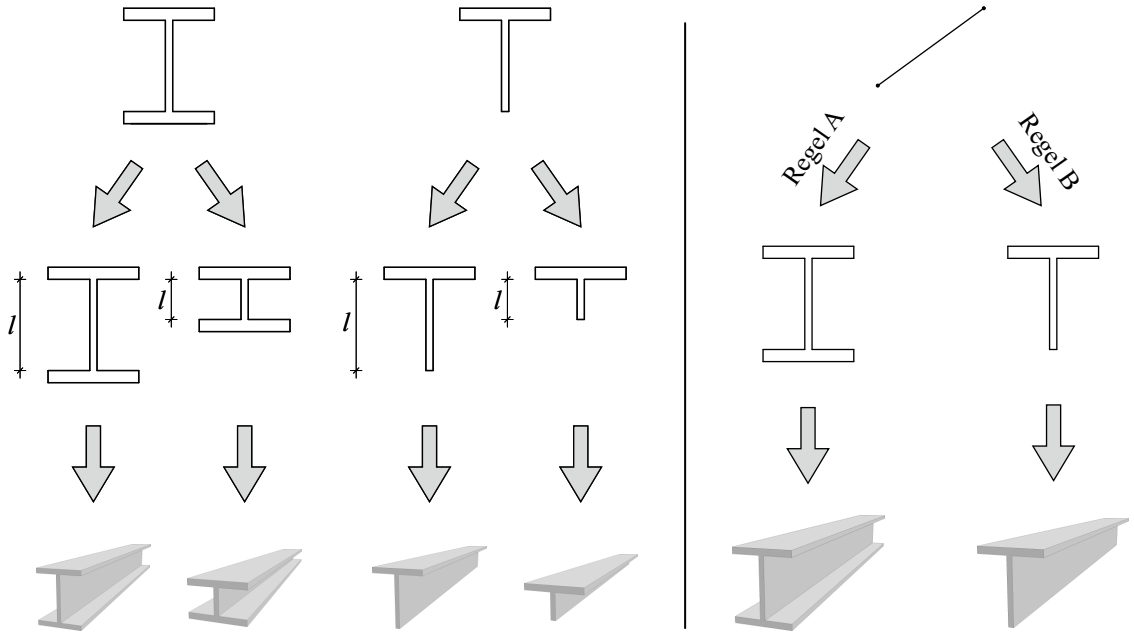


Abbildung 6.2: Links ist dargestellt, wie zwei verschiedene parametrische Modelle auf Basis eines Parameters l verändert werden können. Dies ist die grundlegende *modellvariierende* Funktionalität eines parametrischen Modells, die es aber nicht zulässt, die Art des Profils zu verändern. Durch die Verwendung eines *modellerzeugenden* Graphersetzungssystems, kann zusätzlich sowohl die Erstellung als auch die Änderung der Art des Profils abgebildet werden. Dies ist auf der rechten Seite dargestellt: Dort wird die dem Modell zugrunde liegende Geometrie eines Profils durch die Anwendung zweier Regeln in verschiedenen Varianten erstellt, die auch eine nachträgliche regelbasierte Änderung des Profils zulassen.

Die Modifikation des Zustands eines Systems, also der Übergang eines Zustands in einen anderen Zustand, kann formal und visuell durch Graphersetzungsregeln repräsentiert werden. Damit können Graphersetzungsregeln die durch den Graphen repräsentierte Historie der Modellierungsoperationen verändern. So können verschiedene Modellierungsergebnisse auf Basis einer Modellierungshistorie erzeugt werden, indem Teile der Historie mittels einer Graphersetzungsregel verändert werden. Modelle, die auf diese Weise erstellt wurden, können aufgrund ihres parametrischen Aufbaus aber auch unabhängig von der graphbasierten Repräsentation über die Veränderung von Parametern modifiziert werden. Durch den Einsatz eines Graphersetzungssystems wird diese ohnehin vorhandene Parametrik erweitert. Damit ist das Modell einerseits aufgrund der herkömmlichen genutzten Modellierungstechniken parametrisch und damit in Teilen leicht veränderbar. Zusätzlich kann aber auch der Graph, der das Modell und die Historie der Modellierungsoperationen repräsentiert, verändert werden. So ist es möglich die einem Modell zugrunde liegende herkömmliche Parametrik einerseits zu erzeugen und sogar zu modifizieren, sodass durch das damit veränderte parametrische Verhalten ein neues oder abweichendes Modellierungsergebnis erzeugt werden kann.

Diese Unterscheidung zwischen der herkömmlichen Parametrik eines Modells und der graphbasierten Erzeugung ist in Abbildung 6.2 dargestellt. Auf der linken Seite ist dargestellt, wie ein parametrisches Modell auf Basis eines Parameters l in seinen Abmessungen verändert wird. Dies entspricht der herkömmlichen, einem parametrischen Modell inhärenten Funktionalität, die allein durch die Erstellung eines Modells in einem parametrischen CAD-System gegeben ist. Auf der rechten Seite zeigt die Abbildung zusätzlich, wie ein parametrisches Modell schrittweise durch verschiedene Modellierungsoperationen erstellt werden kann. Diese Modellierungsschritte können durch Graphersetzungsregeln zusammengefasst werden. Wie in der Abbildung dargestellt, ermöglicht dies eine zusätzliche Veränderbarkeit des Modells, da konkurrierende Graphersetzungsregeln[5] definiert werden können, die jeweils zu unterschiedlichen Modellierungsergebnissen führen. Weiterhin kann eine Graphersetzungsregel nicht nur neue Modellierungsoperationen zur Modellierungshistorie hinzufügen, sondern auch bereits erstellte Operationen verändern. So wird die durch parametrische Modellierung mögliche Anpassbarkeit eines Modells selbst automatisiert erstell- und veränderbar.

Ein mithilfe des Graphersetzungssystems erstelltes parametrisches Modell kann also nicht nur durch die Veränderung von Parameterwerten verändert werden. Es ist vielmehr möglich, auch den Erstellungsprozess durch die Anwendung verschiedener Graphersetzungsregeln zu verändern. Diese Möglichkeit ist in Abbildung 6.3 beispielhaft anhand der Schritte zur Modellierung zweier Tunnelquerschnitte dargestellt. Regel A bzw. B erstellt je zwei Skizzen zur Begrenzung der Hülle des Tunnels. Diese werden anschließend durch Regel C extrudiert/gesweeped um den 3D-Körper der Hülle zu erstellen. Unabhängig davon ob

[5] *Konkurrierend* bedeutet hier, dass es für die Detaillierung eines Modells alternative, durch Graphersetzungsregeln definierte Varianten gibt, zwischen denen sich der Nutzer entscheiden kann.

vorher Regel A oder B angewendet wurde, kann nachfolgend Regel D angewendet werden. Durch diese Regel wird auf Basis einer der Skizzen aus dem 1. Schritt (rot) der Boden des Tunnels im Modell erstellt (durch Erstellung und Extrusion/Sweep einer weiteren Skizze). Damit wird das parametrische Modell eines Tunnels erzeugt, das z. B. über die manuelle Änderung des Parameters x angepasst werden kann. Über eine weitere Regel E, die im Graphen das Resultat von Regel A durch das Resultat von Regel B ersetzt, kann zusätzlich die Modellierungshistorie insofern verändert werden, als dass sich der Tunnelquerschnitt verändert, aber die nachfolgende Erstellung des Bodens erhalten bleibt.

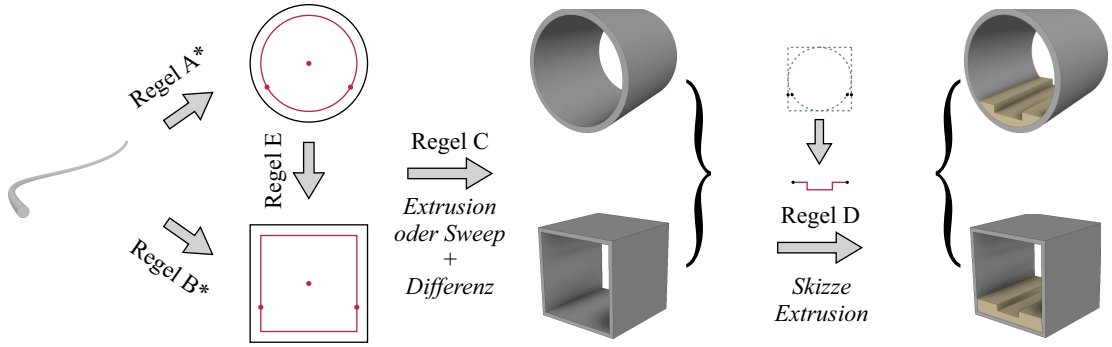


Abbildung 6.3: Durch Graphersetzungsregeln können die dargestellten Schritte zur Modellierung der Hülle zweier Tunnelquerschnitte beschrieben werden. Die Regeln A und B erzeugen unterschiedliche Modellierungsszenarien, die zu verschiedenen Modellen führen. Zusätzlich kann im Nachhinein durch Regel E diese vorherige Entscheidung verändert werden.

6.1.2 Graph und Modell

Wenngleich der wesentliche Zweck des Graphersetzungssystems die formale Beschreibung von Modellierungsabläufen ist, dient der Graph selbst nur dazu, den Zustand eines Modells abzubilden. Die Definition von Graphersetzungsregeln für verschiedene Anwendungsszenarien kann allerdings erst erfolgen, wenn festgelegt wurde, wie ein Graph ein Modell repräsentiert.

Die Graphersetzungsregeln beschreiben also Operationen zur Veränderung, Erweiterung oder Detaillierung des Aufbaus eines parametrischen Modells, während der Graph einen bestimmten Zustand des parametrischen Modells beschreibt. Um ein solches Modell manuell anzupassen oder für nachgelagerte Anwendungsfälle (Ableitung von Plänen, Visualisierung etc.) zu nutzen, muss es möglich sein, ein konkretes parametrisches Modell aus einem Graphen abzuleiten. Ein aus der graphbasierten Repräsentation abgeleitetes parametrisches Modell wird hier als *evaluiertes Modell* bezeichnet. Der Prozess der Ableitung ist die *Interpretation des Graphen*. Durch die Interpretation des Graphen kann also ein evaluiertes Modell erstellt werden.

Um eine eindeutige Interpretation des Graphen zu ermöglichen, müssen neben der formalen Definition des Graphen auch die Konventionen, die bei der Interpretation des Graphen gelten, definiert werden. Damit wird festgelegt, wie die in Kapitel 5 definierten Anforderungen durch den Graphen repräsentiert werden. Die entsprechenden Definitionen werden in den folgenden Abschnitten schrittweise erarbeitet.

Dieses Vorgehen soll hier einleitend veranschaulicht werden. In Abbildung 6.4 ist dazu das parametrische Modell eines Quaders mit quadratischer Grundfläche abgebildet, dessen Abmessungen (Seitenlänge der Grundfläche $d1$, Höhe der Extrusion h) über Parameter verändert werden können. Zusätzlich sind die zur Repräsentation des Modells notwendigen Objekte anhand der zur Modellierung des Quaders durchgeführten Operationen dargestellt. Nicht visualisiert ist dabei, dass der durch den Quader definierte Volumenkörper zu einem Bauteil gehört und dieses Bauteil wiederum Teil einer übergeordneten Baugruppe ist – wenngleich diese Informationen natürlich durch den Graphen repräsentiert werden müssen.

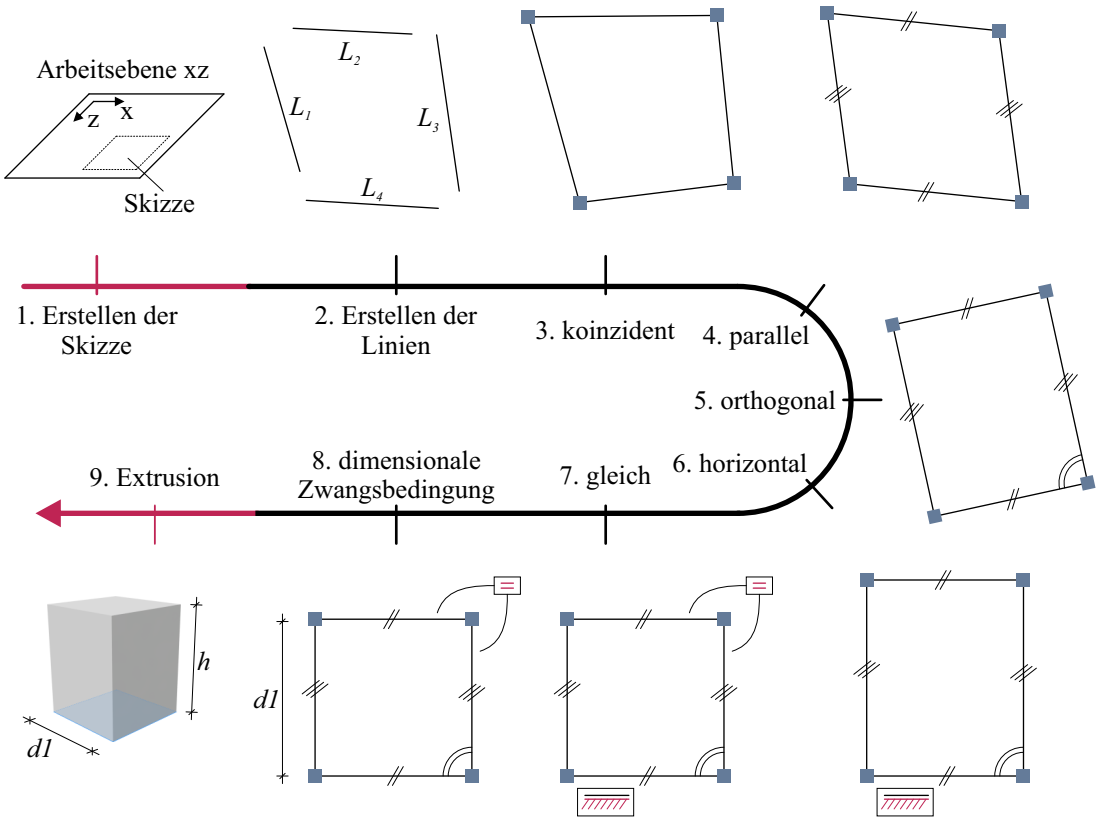


Abbildung 6.4: Ablauf der Modellierungsoperationen mit denen ein Bauteil erstellt wird, dessen Geometrie durch einen quaderförmigen Volumenkörper definiert ist. Der 1. und der 9. Schritt (rot eingefärbt) ist dabei konzeptionell der Konstruktionshistorie auf Bauteilebene zugeordnet, die Schritte 2 bis 8 werden auf Skizzenebene durchgeführt.

Ein Graph, der dieses Modell repräsentiert, muss also Knoten und/oder Kanten zur Abbildung der Baugruppe und des Bauteils enthalten. Die Grundfläche des Quaders wird entsprechend der Abbildung als Skizze auf einer Arbeitsebene (xz-Ebene) des Bauteils

erstellt. Auch Arbeitsebene und Skizze müssen durch den Graphen repräsentiert werden. Zusätzlich muss der Graph reflektieren, wie Baugruppe, Bauteil, Arbeitsebene und Skizze zusammenhängen. In Abbildung 6.5 ist der Graph dargestellt, der diese Zusammenhänge über Knoten und Kanten repräsentiert. Es handelt sich hierbei jedoch noch nicht um eine vollständige Repräsentation, da die Attribute der Knoten und Kanten nicht dargestellt sind. In dieser konzeptionellen Darstellung zur Veranschaulichung des Prinzips sind außerdem der Graph, der die geometrischen Elemente und parametrischen Zwangsbedingungen der Skizze repräsentiert, und der Graph, der die Skizze mit Bauteil und Extrusion verknüpft, getrennt dargestellt.

Die Grundfläche des Quaders wird über die eine parametrische 2D-Skizze eines Quadrats festgelegt (Abbildung 6.4). Diese Skizze setzt sich aus vier Linien (L_1, L_2, L_3, L_4) zusammen. Die Start- bzw. Endpunkte dieser Linien, die an den Ecken aufeinandertreffen, sind koinzident, um sicherzustellen, dass das Polygon immer geschlossen bleibt. Dazu sind vier Zwangsbedingungen vom Typ *koinzident* notwendig. Diese Linien sind paarweise parallel zueinander, was durch zwei entsprechende geometrische Zwangsbedingungen *parallel* festgelegt wird. Weiterhin müssen zwei der nicht parallelen Linien orthogonal zueinander sein. Auch dies wird durch eine Zwangsbedingung definiert. Die Orthogonalität der beiden anderen Linien ergibt sich implizit.

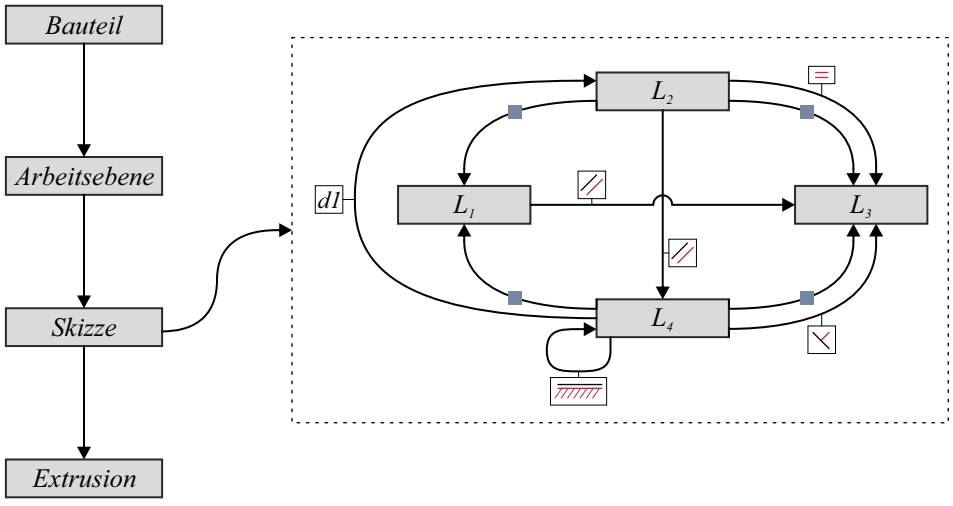


Abbildung 6.5: Graphbasierte Repräsentation der in Abbildung 6.4 dargestellten Modellierungsschritte. Die Zuordnung der Linien L1 bis L4 (und dadurch auch der zugehörigen Zwangsbedingungen) wird hier der Übersichtlichkeit halber durch eine Kante, die auf den gesamten Skizzeninhalt verweist dargestellt. Tatsächlich sind vier Kanten, die von der Skizze auf alle geometrischen Elemente verweisen, notwendig.

Durch die aufgeführten Zwangsbedingungen ergibt sich außerdem, dass die jeweils parallelen Linien die gleiche Länge haben, da die Skizze immer ein Rechteck darstellt. Um die Längen aller Linien gleichzusetzen und damit immer eine quadratische Form zu erzeugen, muss nun

nur noch festgelegt werden, dass zwei Linien, die orthogonal zueinander sind, die gleiche Länge haben. Dies erfolgt mithilfe der geometrischen Zwangsbedingung *gleich*. Damit ist die Geometrie der Skizze so weit bestimmt, dass immer ein quadratischer Polygonzug abgebildet wird. Die Skizze ist aber noch nicht vollständig bestimmt, da es noch möglich ist, das Quadrat zu drehen, zu verschieben oder die Seitenlänge parameterunabhängig zu verändern. Die Position des Quadrats in der Skizze und eine mögliche Drehung werden im Zuge dieses Beispiels allerdings nicht durch Zwangsbedingungen festgehalten, sondern bleiben veränderbar.

Die Seitenlänge wird nun über eine dimensionale Zwangsbedingung fixiert. Eine Änderung der Seitenlänge ist somit nur noch möglich, wenn der Wert des Parameters der dimensionalen Zwangsbedingung verändert wird. Der initiale Wert dieses Parameters entspricht dabei der Seitenlänge des Quadrats zum Zeitpunkt der Erstellung der Zwangsbedingung. Es kann jedoch auch im Zuge der Erstellung ein beliebiger Wert für den Parameter vergeben werden, sodass die Seitenlänge des Quadrats unmittelbar angepasst wird. Damit ist die Modellierung der Skizze abgeschlossen. Die geometrischen Elemente und die verschiedenen Zwangsbedingungen werden entsprechend der Abbildungen als Knoten und Kanten im Graphen erstellt.

In einem letzten Schritt muss nun noch das skizzierte Quadrat extrudiert werden. Hierbei handelt es sich um eine Modellierungsoperation, die nicht mehr im Kontext der Skizze durchgeführt wird, sondern um eine prozedurale Operation auf Ebene des Bauteils. Auch diese Modellierungsoperation wird dem Graphen hinzugefügt und mit der zuvor erstellten Skizze verknüpft. Die Länge der Extrusion muss im Graphen als Parameter, der mit einem Wert versehen ist, hinterlegt sein.

In Abbildung 6.5 ist der Graph dargestellt, der zur Repräsentation dieses einfachen parametrischen Modells notwendig ist. Dieser Graphrepräsentation liegen folgende Konventionen zugrunde:

- Die Knoten und Kanten des Graphen haben verschiedene vordefinierte Typen.
- Die Knoten und Kanten des Graphen haben je nach Typ bestimmte Attribute.
- Die Knoten des Graphen repräsentieren unter anderem geometrische Elemente, Baugruppen, Bauteile, Arbeitsebenen, Skizzen und Extrusionen.
- Die Kanten des Graphen repräsentieren parametrische Zwangsbedingungen und die Beziehungen der durch Knoten dargestellten Objekte zueinander.
- Die Kanten des Graphen sind gerichtet und verbinden immer genau zwei Knoten des Graphen miteinander.

Im folgenden Abschnitt wird beschrieben, wodurch sich diese Konventionen bedingen, also weshalb der Graph genau entsprechend dieser Konventionen aufgebaut wird, um ein parametrisches Modell zu repräsentieren. So kann der Graph schließlich auch formal beschrieben werden und anschließend das Metamodell des Graphen definiert werden.

6.2 Konzeption des Graphen

Um parametrische Modelle mithilfe eines Graphersetzungssystem zu erzeugen, muss in einem ersten Schritt sichergestellt werden, dass diese Modelle überhaupt durch einen Graphen hinsichtlich der im vorangegangenen Kapitel beschriebenen Anforderungen repräsentiert werden können. Dazu müssen diese modellbezogen formulierten Anforderungen so formalisiert werden, dass sie in einem Graphen abgebildet werden können.

Die größte Herausforderung besteht dabei darin, den möglichen allgemeinen Aufbau des Graphen so zu definieren und dabei auch einzuschränken, dass allein dadurch schon weitestgehend sichergestellt ist, dass ein bestimmter Graph auch ein valides parametrisches Modell repräsentiert. Würde der Graph uneingeschränkt durch eine beliebige Kombination von Knoten und Kanten verschiedener Typen erstellt werden können, würde die Interpretation des Graphen unweigerlich in vielen Fällen nicht zu einem evaluierbaren Modell führen, da der Graph letztlich ein in der Realität unmögliches Modell beschreibt. Dies kann beispielsweise damit verglichen werden, dass bei der Beschreibung der Oberfläche geometrischer Körper mittels Drahtmodellen sogenannte nonsense-Objekte erzeugt werden können.

Die Methodik, die sich im Rahmen dieser Arbeit als zielführend zur Entwicklung des Graphen und damit auch des Graphersetzungssystems herausgestellt hat, ist ein iteratives Vorgehen. Dabei wird der abstrakte Graph und die entsprechenden Graphersetzungsregeln konsequent zusammen mit den durch sie repräsentierten Modellen und Modellierungsschritten in einem parametrischen CAD-System betrachtet. Zusätzlich wird auch immer die Software, die zur Modellierung des Graphersetzunggsystems genutzt wird, einbezogen, damit eine Demonstration der praktischen Anwendung der entwickelten Methodik möglich ist[6]. Alle Definitionen, die sich aus der Umsetzung der Anforderungen, welche sich wiederum aus dem Konzept der parametrischen Modellierung ergeben, begründen, müssen im Kontext dieser drei Bereiche betrachtet werden. Jede Änderung an der formalen Definition des Graphen und der Graphersetzungsregeln muss dabei hinsichtlich der Modellierung und der Interpretation des Graphen berücksichtigt und iterativ angepasst werden. Nur durch dieses iterative Vorgehen kann letztlich ein durchgängiges Konzept entwickelt werden, mit dem die korrekte Funktionalität des Graphersetzungssystems sichergestellt ist.

[6]Hierbei ist anzumerken, dass durch das verwendete Tool zur Modellierung des Graphersetzunggsystems (GrGen.NET) keinerlei Einschränkungen in Kauf genommen werden mussten.

In den folgenden Abschnitten werden die verschiedenen Aspekte der Konzeption des Graphen beschrieben, die sich aus diesem iterativen Vorgehen ergeben haben. Dabei werden auch alternative Ansätze, die im Zuge des iterativen Vorgehens verworfen wurden, kurz beschrieben.

6.2.1 Eigenschaften des Graphen

In Kapitel 4 wurden verschiedene Typen von Graphen vorgestellt. Der Typ des Graphen ergibt sich aus den Attributen des Graphen. Um einen Graphen zu konzipieren, der für die Anwendung im Kontext einer bestimmten Problemstellung angewendet werden soll, muss der Typ des Graphen festgelegt werden. Die *Attribute des Graphen*, durch die sich der *Typ des Graphen* definiert, beziehen sich dabei auf den Graphen selbst. Sie sind nicht mit den Typen und Attributen der Knoten und Kanten des Graphen zu verwechseln.

Diese Festlegung muss dabei nicht zwangsläufig direkt zu Beginn des Entwicklungsprozesses final festgelegt werden. Vielmehr erscheint es sinnvoll, die Möglichkeiten, die sich durch die verschiedenen Typen von Graphen ergeben, vollumfänglich zu betrachten, und so den optimalen Typ für einen bestimmten Anwendungsfall zu identifizieren. Optimal bedeutet in diesem Fall, dass dem Graphen ausgehend von einem einfachen Graphen so lange weitere Attribute hinzugefügt werden, bis durch den Graph alle Anforderungen eines Anwendungsfalls berücksichtigt sind. Offensichtlich sollten hierbei nur tatsächlich notwendige Attribute hinzugefügt werden. Die Notwendigkeit für bestimmte Attribute des Graphen ergibt sich zuvorderst aus den Anforderungen an die graphbasierte Repräsentation.

Weiterhin wurde bei der Konzeption des Graphen aber auch konsequent berücksichtigt, dass die graphbasierte Repräsentation möglichst übersichtlich ist und eine gute Lesbarkeit gewährleistet wird. Auch wenn der Graph nicht durch einen Anwender des Graphersetzungssystems manuell interpretiert werden muss, ist dies bei der Erstellung von Graphersetzungsregeln zwingend notwendig. Um die ohnehin anspruchsvolle Tätigkeit des Erstellens von Graphersetzungsregeln zu unterstützen, wurde daher bei der Konzeption des Graphen darauf geachtet, eine möglichst anschauliche Repräsentation zu gewährleisten, wenn dies unter Berücksichtigung der Anforderungen möglich war.

Bei dem hier vorliegenden Anwendungsfall einer graphbasierten Repräsentation hat sich bereits sehr früh herausgestellt, dass ein *einfacher Graph* nicht praktikabel ist, um ein parametrisches Modell vollständig zu repräsentieren. Theoretisch ist es zwar beispielsweise möglich, die Objekte, aus denen sich das parametrische Modell zusammensetzt, durch Subgraphen zu repräsentieren, über deren topologische Struktur sich der Typ des Objekts ableiten lässt. Allerdings würde der Graph dadurch keine annähernd anschauliche Repräsentation des parametrischen Modells darstellen und zusätzlich würde die Erstellung von Graphersetzungsregeln unnötig erschwert. Daher ist es sinnvoll, einen getypten (oder

auch benannten) Graphen zu verwenden, bei dem anhand des Typs eines Knotens oder einer Kante ableitbar (und auch gut ablesbar) ist, welches Objekt in einem parametrischen Modell von diesem Knoten oder dieser Kante repräsentiert wird. So wird beispielsweise ein Linien-Objekt im parametrischen Modell durch einen Knoten des Typs Linie *(line)* repräsentiert.

Zusätzlich ist es notwendig, den Knoten und Kanten Attribute zuweisen zu können. Insofern muss ein *getypter attributierter Graph* verwendet werden. Die Attribute sind notwendig, um die Eigenschaften der Objekte im parametrischen Modell direkt mit dem repräsentierenden Element im Graphen abbilden zu können. Auch hier wäre es prinzipiell möglich, stattdessen zusätzliche Knoten und Kanten zu verwenden, durch die die Eigenschaften und die Werte der Eigenschaften definiert werden. Allerdings müsste dann eine Vielzahl zusätzlicher Typen von Knoten zur Repräsentation der Eigenschaften definiert werden. Da auch diese Knoten keine Attribute hätten, müssten die Werte der Eigenschaften über weitere Knoten dargestellt werden, deren Typ den Wert einer Eigenschaft repräsentiert. Geht man davon aus, dass die Werte der Eigenschaften (z. B. die Größe einer temporären Koordinate) beliebige Werte annehmen können, wäre es nicht mehr realistisch möglich, die Menge der verschiedenen Knotentypen abschließend zu definieren. Auch hier würde die Anschaulichkeit der Darstellung verringert und die Erstellung von Graphersetzungsregeln, die – wie später beschrieben – weitgehend manuell erfolgt, deutlich erschwert.

Durch die Verwendung von gerichteten Kanten kann bei allen Kanten zwischen dem Anfangs- und dem Endknoten unterschieden werden. Damit ist beispielsweise die Reihenfolge, in der die Knoten eines Graphen interpretiert werden, steuerbar. Außerdem werden dadurch Abhängigkeiten der Knoten untereinander veranschaulicht, wie beispielsweise, dass eine Skizze auf einer bestimmten Arbeitsebene liegt. Weiterhin ermöglicht die Verwendung von gerichteten Kanten das Herstellen eines Bezugs zwischen den Attributen einer Kante und den Knoten, die durch sie verbunden werden. Dies ist für die in Abschnitt 6.2.4 beschriebenen Andockstellen von durch den Graphen repräsentierten Objekten relevant, da diese sich jeweils auf einen der beiden Knoten, die eine Kante verbinden, beziehen. Durch die Richtung der Kante wird hier festgelegt, auf welchen der beiden inzidenten Knoten sich die durch ein Attribut definierte Andockstelle bezieht.

Weiterhin muss der Graph parallele Kanten enthalten können. Parallele Kanten müssen dabei nicht in die gleiche Richtung zeigen. Sie müssen auch nicht vom gleichen Typ sein, da sich in vielen Fällen die Notwendigkeit ergibt, zwei Knoten durch mehrere Kanten unterschiedlichen Typs zu verbinden. Dies ist beispielsweise immer dann der Fall, wenn zwei geometrische Elemente durch mehr als nur eine parametrische Zwangsbedingung miteinander verknüpft sind.

Durch die parametrische Zwangsbedingung ergibt sich auch die Notwendigkeit, dass der Graph Schleifen enthalten kann. Wie bereits in Kapitel 4 erläutert wurde, handelt es sich

bei Schleifen um Kanten, deren Anfangs- und Endknoten identisch sind. Beispielsweise beziehen sich die geometrischen Zwangsbedingungen *horizontal* oder *vertika*l teils nur auf ein geometrisches Element, sodass der Anfangs- und Endknoten der entsprechenden Kante dieses Element repräsentieren muss.

Bei den in dieser Arbeit verwendeten Graphen zur Repräsentation von parametrischen Modellen handelt es sich damit um *attributierte getypte Multidigraphen*. Zusammengefasst bedeutet dies:

- Die Kanten des Graphen sind hinsichtlich ihrer Richtung definiert. Es ist also für jede Kante ein Anfangs- und ein Endknoten festgelegt. Dies wird in den Abbildungen visualisiert, indem die Kanten als Pfeile, die vom Anfangs- zum Endknoten zeigen, dargestellt sind.
- Der Graph darf Schleifen enthalten. Eine gerichtete Kante kann also denselben Knoten als Anfangs- und ein Endknoten besitzen.
- Zwei Knoten des Graphen können durch mehr als eine Kante bzw. mit einer beliebigen Anzahl an Kanten miteinander verbunden sein.
- Die Knoten und Kanten des Graphen sind getypt. Je nach Typ eines Knotens oder einer Kante bestimmt sich, welche Attribute ein Knoten oder eine Kante aufweist.
- Die Knoten und Kanten des Graphen sind attributiert. Ein Knoten oder eine Kante kann also eine beliebige Anzahl von Attributen aufweisen. Den Attributen sind Werte zugeordnet.

6.2.2 Hierarchischer Aufbau des Graphen

Der Aufbau eines parametrischen Modells, das mit einer gängigen CAD-Anwendung erstellt wurde, wird im Wesentlichen durch die Verwendung von Baugruppen, Bauteilen, und Skizzen definiert. Die Modellierungsoperationen, die ein Nutzer in einer CAD-Anwendung zur Verfügung hat, sind davon abhängig, ob auf Baugruppen-, Bauteil-, oder Skizzenebene modelliert wird. Es kann insofern von einem hierarchischen Aufbau des parametrischen Modells gesprochen werden, der entsprechend dieser Ebenen definiert ist.

Modellierungsoperationen sind implizit einer dieser Ebenen zugeordnet, was sich auch bei der manuellen Modellierung direkt durch die Werkzeuge, die dem Anwender in einem parametrischen CAD-System zur Verfügung stehen, widerspiegelt. Visualisiert man die Schritte eines Modellierungsprozesses und deren Abfolge und Abhängigkeiten, so erhält man eine baumartige Struktur..

Unter Berücksichtigung dieser Gegebenheiten hat es sich als sinnvoll erwiesen, zwischen drei verschiedene Ebenen der Repräsentation zu unterscheiden, die sich aus dem beschriebenen hierarchischen Aufbau parametrischer Modelle ergeben:

- Die unterste Ebene beschreibt Skizzen, also 2D-Zeichnungen mit parametrisierten Abmessungen und definierten geometrischen Zwangsbedingungen, die als Grundlage für nachfolgende Modellierungsoperationen zur Erstellung von 3D-Körpern in Bauteilen dienen.
- Der Aufbau eines Bauteils und die Erzeugung der 3D-Körper wird in der zweiten Ebene abgebildet. Diese Ebene beinhaltet auch die Reihenfolge der Konstruktionsoperationen sowie deren Abhängigkeiten untereinander.
- In der dritten Ebene wird beschrieben, wie Baugruppen auf Basis mehrerer Bauteile zusammengesetzt und mithilfe von Mating-Zwangsbedingungen aneinander ausgerichtet werden können.

Diese drei Ebenen werden im Folgenden als Baugruppen-, Bauteil- und Skizzengraph bezeichnet.

Es stellt sich nun die Frage, wie die Ebenen und damit die Struktur des parametrischen Modells in einer graphbasierten Repräsentation abgebildet werden können. Auf jeder Ebene müssen unterschiedliche Objekte, die in verschiedenen Verbindungen zueinanderstehen, betrachtet werden. Da sich die Modellierungsoperationen je nach Ebene unterscheiden, ist es dabei sogar denkbar, getrennte Graphersetzungssystem zu verwenden, die in sich abgeschlossen sind und sich nur auf jeweils eine Ebene der graphbasierten Repräsentation beziehen.

Beispielsweise werden beim Erstellen einer Skizze nur bestimmte geometrische Elemente und parametrische Zwangsbedingungen genutzt, die auf Bauteil- und Baugruppenebene nicht relevant sind. Innerhalb der Skizze können hingegen keine dieser Objekte modelliert werden. Auch die Reihenfolge, in der die Objekte in einer Skizze erstellt wurden, spielt nach Fertigstellung der Skizze keine Rolle für die weitere Verwendung der Skizze im Modellierungsprozess mehr. Es ist dementsprechend denkbar, ein Graphersetzungssystem zu konzipieren, das nur die Schritte der prozeduralen Modellierung auf Bauteilebene abbildet und in dem eine Skizze als einzelner Knoten repräsentiert wird. Die Erstellung und Veränderung von Skizzen würde dann in einem eigenständigen Graphersetzungssystem abgebildet[7]. Die durch dieses Graphersetzungssystem erstellten Graphen werden dann in die Skizzenknoten des Graphen zur Repräsentation eines Bauteils eingebunden.

[7]Damit würden innerhalb des Graphersetzungssystems für Skizzen nur der variationale Modellierungsansatz angewendet werden, während auf Bauteilebene der prozedural-parametrische Modellierungsansatz relevant ist.

Bei der Übertragung des hierarchischen Aufbaus eines parametrischen Modells in eine graphbasierte Repräsentation als Grundlage für ein Graphersetzungssystem sind also zwei verschiedene Varianten denkbar:

1. Nutzung von übergeordneten Graphen, die die Struktur des Modells abbildet und Verweise auf weitere Graphen enthält, die in sich abgeschlossene Teile des Modells repräsentieren.

2. Integration aller Objekte und Informationen in einen allumfassenden Graphen. Graphen, die ein Bauteil beschreiben, sind dabei Subgraphen des Graphen zur Beschreibung von Baugruppen, während Graphen, die eine Skizze repräsentieren, Subgraphen des Bauteilgraphen sind.

Wird angenommen, dass getrennte Graphersetzungssysteme für die Repräsentation des Modells notwendig sind, ist die naheliegendste Option ein Verweis innerhalb eines Knotens im Baugruppengraphen auf einen extern gespeicherten Bauteilgraphen bzw. innerhalb eines Knotens im Bauteilgraphen auf einen extern gespeicherten Skizzengraphen. Für jede im Rahmen des Modellierungsprozesses erstellte Skizze wird dabei ein eigener Skizzengraph angelegt. Dieser ist in sich geschlossen und es können keine durch Kanten modellierte Beziehungen zwischen den Elementen verschiedener Skizzengraphen bestehen.

Weiterhin ist es nicht möglich, Knoten innerhalb des Skizzengraphen mit Knoten der anderen beiden Graphen zu verbinden. Dies gilt analog für Bauteilgraphen. Graphersetzungsregeln, die auf einen Bauteil, Baugruppen- oder einen der Skizzengraphen angewendet werden, können also keine direkten Auswirkungen auf die jeweils anderen Graphen haben, sondern müssten die Ausführung weiterer Graphersetzungsregeln auslösen, um gegebenenfalls notwendige Änderungen in einen der anderen Graphen zu propagieren. Die einzige Verbindung zwischen den verschiedenen Graphen besteht letztlich in den Verweisen in den Knoten eines übergeordneten Graphen (beispielsweise des Bauteilgraphen) in den externen Graphen auf der niedrigeren Ebene (beispielsweise den Skizzengraphen). In diesen Knoten wird eindeutig definiert, auf welchen weiteren Graphen jeweils verwiesen wird.

Demgegenüber steht der Ansatz, ein einziges übergreifendes Graphersetzungssystem für alle drei Ebenen zu verwenden. Das Metamodell eines solchen Systems muss somit alle Knoten- und Kantentypen, die im Baugruppen-, Bauteil- und Skizzengraph enthalten sein können, beinhalten. Dieser Lösungsansatz hat den wesentlichen Vorteil, dass eine einzelne Graphersetzungsregel direkte Auswirkungen auf alle Repräsentationen der Objekte eines parametrischen Modells haben kann. Zusätzlich sind in einem solchen Graphen Beziehungen in Form von Kanten zwischen allen Knotentypen umsetzbar. Es können also beispielsweise direkt Beziehungen zwischen den geometrischen Elementen unterschiedlicher Skizzen repräsentiert werden.

Die Entscheidung, welche der beiden Varianten zur Strukturierung des Graphen zu bevorzugen ist, ergibt sich aus der Betrachtung der Zusammenhänge zwischen den Objekten auf verschiedenen Ebenen. Generell existiert für ein Modell genau ein Baugruppengraph. Teile eines Modells, die in diesem Baugruppengraphen repräsentiert werden, erfordern die Erstellung von Bauteilen, für die wiederum die Erstellung von Skizzen notwendig ist. Anders ausgedrückt ist das Erstellen einer Skizze eine der Modellierungsoperationen, die im Bauteilgraph repräsentierbar sein müssen. Um eine Skizze sinnvoll in einem Bauteil nutzen und platzieren zu können, muss es möglich sein, nicht nur die Skizze als Ganzes einzubinden, sondern auch Beziehungen zwischen den geometrischen Elementen der Skizze und den Objekten auf Bauteilebene definieren zu können. Außerdem müssen Beziehungen zwischen den geometrischen Elementen unterschiedlicher Skizzen abgebildet werden können, um Abhängigkeiten, die zwischen den Elementen verschiedener Skizzen vorhanden sind, abzubilden. Nur so kann eine skizzenübergreifende Parametrik im Modell gewährleistet werden, wie sie durch die Nutzung projizierter Geometrie (geometrische Elemente werden von einer Skizze in eine andere Skizze projiziert) möglich ist. Auch zur Nutzung von Mating-Zwangsbedingungen muss es möglich sein, die geometrischen Elemente unterschiedlicher Skizzen miteinander zu verknüpfen. Demzufolge ist es erforderlich, Graphersetzungsregeln zu definieren, die sich auf Elemente verschiedener Ebenen beziehen.

Bei der Gegenüberstellung der Varianten unter Berücksichtigung dieser Aspekte zeigt sich schnell, dass sich die Integration der gesamten Repräsentation in einem Graphen als am zielführendsten erweist. Auch wenn die Graphen bei der Wahl dieser Variante vergleichsweise umfangreich werden können und eine Visualisierung bei komplexeren Modellen dementsprechend unübersichtlich wird[8], überwiegen die Vorteile. Weiterhin existieren verschiedene Möglichkeiten, mit denen auch komplexe Graphen so gezeichnet werden können, dass die Darstellung so übersichtlich wie möglich bleibt. Dazu können sowohl Teile des Graphen gruppiert als auch unterschiedliche Algorithmen zum Zeichnen des Graphen gewählt werden. Verschiedene Varianten sind in Abschnitt 4.1.2 dargestellt. Die Abbildung 4.8 zeigt deutlich, dass ein Graph der hierarchisch gezeichnet wurde und bei dem es möglich ist, Knoten zu gruppieren, eine übersichtliche Variante darstellt.

Bei der gewählten Variante findet sich die Struktur des parametrischen Modells trotzdem in der graphbasierten Repräsentation wieder. Dies bedingt sich auch dadurch, dass die Abhängigkeiten und damit die Reihenfolge der Modellierungsschritte, die bei der Interpretation eines Graphen zur Modellerzeugung ausgeführt werden müssen, weitgehend diesem hierarchischen Aufbau entsprechen. Der gesamte Graph ist in diesem Fall allerdings kein Baum mehr, da zusätzliche Zusammenhänge zwischen Elementen, die sich auf unterschiedlichen Ebenen in unterschiedlichen Ästen befinden, Teil einer vollständigen Repräsentation sein müssen.

[8] Dies ist allerdings für die praktische Anwendung nur bedingt relevant, da die erzeugten Graphen in der Regel nicht visuell interpretiert werden müssen.

Die Abbildung der Hierarchie des parametrischen Modells im Graphen wird über die Zuordnung der verschiedenen Knotentypen zu einer dieser Ebenen realisiert. Somit ist jeder Knoten entsprechend seinem Typ eindeutig einer Ebene zugeordnet. Die Zuordnung der Knotentypen zu den Ebenen ergibt sich dabei intuitiv aus dem Aufbau eines parametrischen Modells entsprechend den Objekten, die durch Knoten dieses Typs repräsentiert werden.

Die Zugehörigkeit eines Kantentyps zu einer Ebene kann hingegen nicht direkt aus den durch sie repräsentierten Objekten abgeleitet werden. Der Grund hierfür ist, dass nicht alle Kanten zur Repräsentation eines konkreten Objekts genutzt werden, sondern nur dazu, die Zusammenhänge zwischen den Objekten zu repräsentieren. Die Zuordnung der Kantentypen bedarf dementsprechend zusätzlicher Festlegungen. Verbinden Kanten ausschließlich Knoten, die auf der gleichen Ebene liegen, so wird auch der entsprechende Kantentyp dieser Ebene zugeordnet. Bei Kanten, die zur Repräsentation von ebenenübergreifenden Zusammenhängen genutzt werden, wird die Kante jeweils der Ebene zugeordnet, die in der Hierarchie höher steht.

Die Zuordnung der verschiedenen Knoten- und Kantentypen zu den einzelnen Ebenen des Graphen wird bei der Beschreibung der Typen vorgenommen.

6.2.3 Verwendung von Knoten und Kanten

Eine wesentliche Entscheidung, die bei der Konzeption einer graphbasierten Repräsentation getroffen werden muss, besteht in der Festlegung *was genau* durch die Knoten und Kanten des Graphen jeweils repräsentiert werden soll. In diesem Abschnitt wird beschrieben, welche Objekte eines parametrischen Modells durch Knoten und welche Objekte durch Kanten repräsentiert werden und wodurch sich genau dieses Vorgehen bedingt.

In diesem Kontext muss auch berücksichtigt werden, dass bei der Verwendung eines Multidigraphen eine Kante maximal zwei Knoten miteinander verbinden kann. Beispielhaft lässt sich dies an der graphbasierten Repräsentation eines Systems aus Wegen und Kreuzungen, wie es in Abb. 6.6 (links) dargestellt ist, beschreiben. In einer graphbasierten Repräsentation werden meist intuitiv die Kreuzungen durch Knoten und die Wege durch Kanten repräsentiert (Mitte). Dass dies aber nicht zwangsläufig der Fall sein muss, zeigt der rechte Teil der Abbildung. Hier ist das gleiche System durch einen Hypergraphen (siehe Kapitel 4) repräsentiert, bei dem eine Kante mehr als zwei Knoten verbinden kann. Nur durch die Verwendung eines Hypergraphen ist es in diesem Beispiel also überhaupt möglich, frei zu entscheiden was die Knoten und Kanten jeweils repräsentieren.

Prinzipiell beschreiben beide Varianten der graphbasierten Repräsentation in diesem Beispiel formal betrachtet die gleichen Zusammenhänge. Es ist jedoch unschwer zu erkennen, dass die Visualisierung bei der Verwendung eines Hypergraphen weniger intuitiv zugänglich ist

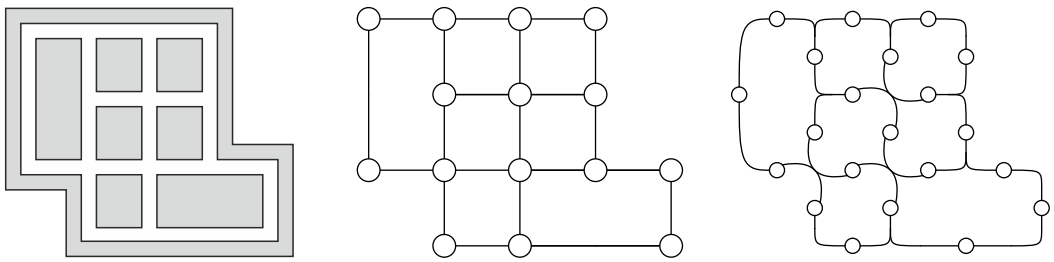


Abbildung 6.6: Darstellung eines Straßennetzes aus Wegen und Kreuzungen (links) und der entsprechenden Repräsentation als einfacher Graph (mitte) und Hypergraph (rechts).

und bei größeren Graphen schnell unübersichtlich werden kann. Im vorliegenden Ansatz wurden Hypergraphen daher zwar nicht von Vornherein ausgeschlossen, letztlich wurde aber auf die Verwendung verzichtet. Dies bedingt sich im Wesentlichen in der fehlenden Notwendigkeit für die Nutzung von Kanten, die mehr als zwei Knoten verbinden. Da die Verwendung von Hyperkanten an keiner Stelle nötig ist, reicht ein Multidigraph aus. In praktischer Hinsicht vereinfacht dies die Erstellung von Graphersetzungsregeln und erhöht die Anzahl der Softwarebibliotheken, die zur Implementierung des Graphersetzungssystems genutzt werden können.

Bei der Konzeption der graphbasierten Repräsentation war es frühzeitig absehbar, dass in den meisten Fällen die Notwendigkeit besteht, ein geometrisches Element durch parametrische Zwangsbedingungen mit mehreren weiteren geometrischen Elementen zu verknüpfen und zusätzlich einer Skizze zuzuordnen. Für eine Zwangsbedingung ist es hingegen ausreichend, wenn sie zwei geometrische Elemente verbindet bzw. nur einem geometrischen Element zugeordnet ist. Demzufolge besitzen Knoten in einem Multidigraph also die notwendigen Voraussetzungen, um geometrische Elemente zu repräsentieren, während sich Kanten gut eignen, um die parametrischen Zwangsbedingungen, die diese Elemente verbinden, abzubilden.

In der hier konzipierten graphbasierten Repräsentation lässt sich die Verwendung von Knoten und Kanten wie folgt zusammenfassen:

- Die folgenden Objekte werden durch Knoten repräsentiert:
    - Objekte die die hierarchische Struktur des Modells definieren (Baugruppen, Bauteile, Skizzen)
    - prozedurale Modellierungsoperationen (Extrusion, Sweep etc.)
    - geometrische Elemente in einer Skizze (Punkte, Linien etc.)
- Die folgenden Objekte und Zusammenhänge werden durch Kanten repräsentiert:
    - Mating Zwangsbedingungen auf Baugruppenebene

- parametrische Zwangsbedingungen auf Skizzenebene
- Abhängigkeiten auf Bauteilebene und damit auch die Reihenfolge der Modellierungsoperationen
- Zuordnung von geometrischen Elementen zu einer Skizze
- Zuordnung von Skizzen zu Bauteilen
- Zuordnung von Bauteilen zu Baugruppen

Die entsprechenden Typen von Knoten und Kanten werden in Abschnitt 6.4 detailliert hinsichtlich ihrer Attribute beschrieben. In Abschnitt 6.2.3 wurde bereits erklärt, wodurch sich die Entscheidung begründet, ein Objekt eines parametrischen Modells mithilfe eines Knotens bzw. einer Kante zu repräsentieren.

6.2.4 Andockstellen

In Kapitel 5, Abschnitt 5.5.1 wurde bereits beschrieben, dass sich bestimmte parametrische Zwangsbedingungen immer auf einen speziellen Teil eines geometrischen Elements – eine sog. Andockstelle – beziehen. Diese Andockstellen sind als Teil der Beschreibung der betrachteten geometrischen Elemente bereits aufgeführt. Sie sind beispielsweise nötig, damit bei der Verknüpfung zweier Linien durch die Zwangsbedingung *koinzident* eindeutig definiert ist, ob die Anfangs-, Mittel- oder Endpunkte der Linie koinzident sein soll. In Abbildung 6.7 sind die verschiedenen Möglichkeiten der Koinzidenz zweier Linien dargestellt. Nicht dargestellt sind die weiteren Varianten, bei denen ein Punkt – entsprechend der Definition der Zwangsbedingung *koinzident* in Abschnitt 5.5.2 – nicht koinzident zu einem anderen Punkt sondern zur Linie als Ganzes ist und damit an einem beliebigen Punkt auf der Linie liegen kann.

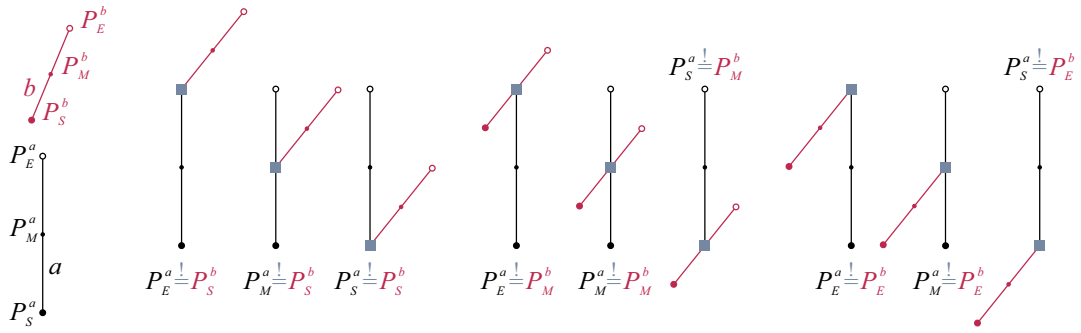


Abbildung 6.7: Die beiden Linien, a und b werden durch die Zwangsbedingung koinzident verknüpft. Falls nicht angegebenen wird, ob jeweils die Start-, Mittel-, oder Endpunkte der Linien zusammenfallen sollen, sind die dargestellten Varianten möglich. Eine Interpretation führt also nicht zu einem eindeutigen Ergebnis.

Das dargestellte Beispiel lässt sich auf die weiteren geometrischen Elemente übertragen. Bei fast allen Kombinationen von zwei geometrischen Elementen, die koinzident zueinander sind, gibt es verschiedene Varianten. Die einzige Ausnahme bildet dabei die Koinzidenz zweier Punkte. Weiterhin sind auch bei einer dimensionalen Zwangsbedingung, die den Abstand zweier Elemente in Abhängigkeit eines Parameters definiert, je nach Wahl der Andockstelle verschiedene Varianten als Ergebnis einer Interpretation möglich.

Die Andockstellen sind daher zur Beschreibung der Topologie einer parametrischen Skizze von elementarer Wichtigkeit und müssen in der graphbasierten Repräsentation abgebildet werden. Andernfalls ist die graphbasierte Repräsentation eines parametrischen Modells nicht eindeutig, sodass die Interpretation des Graphen unterschiedliche evaluierte Modelle ergeben könnte oder nicht fehlerfrei möglich ist.

Es muss daher festgelegt werden, wie die Andockstellen der geometrischen Elemente, auf die sich die parametrischen Zwangsbedingungen beziehen, im Graphen abgebildet werden, um eine topologische Verknüpfung zwischen verschiedenen geometrischen Elementen herzustellen. Da geometrische Elemente als Knoten und parametrische Zwangsbedingungen als Kanten zwischen diesen Knoten repräsentiert werden, gilt es, die Inzidenz dieser Knoten und Kanten verfeinert zu definieren. Entsprechend der in Kapitel 4 eingeführten Grundlagen der Graphentheorie ist dies aber nicht direkt möglich, da ein Knoten und eine Kante entweder inzident sind oder nicht. Auch wenn in gerichteten Graphen zusätzlich unterschieden werden kann, ob ein Knoten der Anfangs- oder der Endknoten einer Kante ist, reicht dies nicht aus, um zu definieren, auf welche Andockstelle eines Knotens sich eine Kante bezieht. Trotzdem sind verschiedene Möglichkeiten denkbar, diesen Zusammenhang zwischen einem geometrischen Element und einer parametrischen Zwangsbedingung graphbasiert zu beschreiben. Diese wirken sich aber entweder auf den grundlegenden Typen eines Graphen, seine Struktur oder auf die Attributierung der Graphelemente aus.

Im Ansatz von Helms (2013) wird ein Graph verwendet, um den Entwurf einer Produktarchitektur zu automatisieren. Die Knoten eines Graphen werden dabei genutzt, um die einzelnen Bestandteile einer Produktarchitektur zu repräsentieren. Auch in diesem Ansatz ist es nötig, für bestimmte Knoten zu definieren, wie genau sie durch eine Kante verbunden sind. Dafür werden sog. *ports* verwendet, die im Wesentlichen die gleiche Funktion haben, wie die hier beschriebenen Andockstellen. Die Problematik wird dort gelöst, indem die *ports* als zusätzliche Knoten im Graphen repräsentiert werden, wobei auch direkt sichergestellt werden kann, dass ein *port* nur inzident zu einer bestimmten Anzahl von Kanten ist. Ein analoges Vorgehen zur Repräsentation der Andockstellen ist in Abbildung 6.8 dargestellt. Hier ist links direkt sichtbar, dass sich die Struktur des Graphen maßgeblich verändert, da eine Vielzahl zusätzlicher Knoten notwendig wird.

In verschiedenen weiteren Ansätzen (Alves *et al.*, 2011; Fernández *et al.*, 2020) werden solche *ports* auch direkt in die mathematisch-formale Definition eines Graphen integriert.

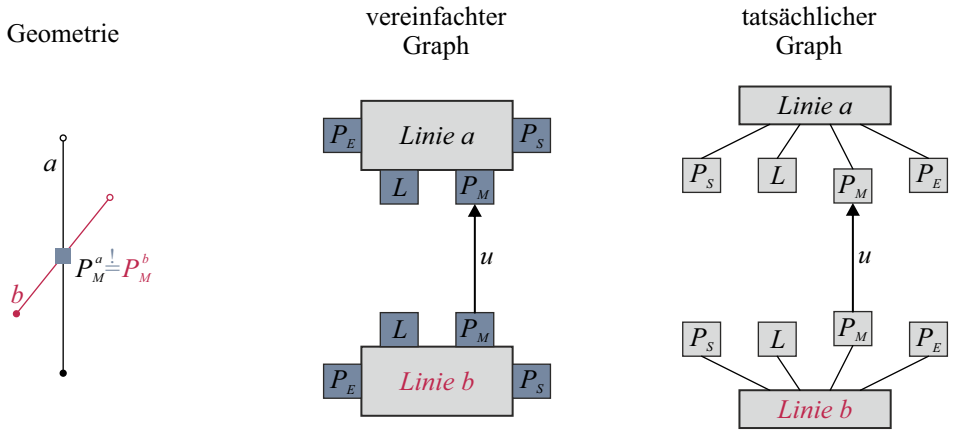


Abbildung 6.8: Nutzung von *ports* zur Definition der Andockstellen, analog zum Ansatz von Helms (2013). Die Kante u repräsentiert die Koinzidenz von P_M^a und P_M^b.

Diese werden dann beispielsweise als *Labeled Multigraphs with Ports* bezeichnet (Andrei *et al.*, 2008). Um solche Graphen mithilfe von Graphersetzungsoperationen zu verändern, ist es allerdings notwendig, dezidierte *port graph rewrite rules* also Graphersetzungsregeln für Graphen mit *ports* zu definieren (Fernández *et al.*, 2014).

Im hier betrachteten Szenario werden in vielen Skizzen allerdings nicht alle Andockstellen der vorhandenen geometrischen Elemente benötigt. Diese erstgenannte Variante führt also dazu, dass in einem Graphen, der eine Skizze als Teil eines parametrischen Modells repräsentiert, in vielen Fällen eine große Anzahl an „freien“ Andockstellen in Form von unnötigen Knoten vorhanden sind. Dies bringt eine unnötige Vergrößerung des Graphen mit sich, die auch den Aufwand der Konzeption von Graphersetzungsregeln erhöhen würde. Es ist für eine Andockstelle außerdem nicht relevant, wie viele Zwangsbedingungen sich auf sie beziehen. Auch die Verwendung eines Graphen mit *ports* bringt keinerlei Vorteile mit sich, die die zusätzlichen notwendigen Definitionen bei der Erstellung von Graphersetzungsregeln rechtfertigen würden.

Daher wird in dieser Arbeit die Information, auf welche Andockstelle sich eine Zwangsbedingung bezieht, stattdessen in den Attributen der Kanten, die diese Zwangsbedingung repräsentieren, definiert. Durch die Richtung der Kante kann zwischen den beiden Knoten, die sie verbindet, unterschieden werden. Es muss daher jeweils ein Attribut definiert werden, durch das sich die Andockstelle des Anfangsknotens und die des Endknotens festlegen lässt. Diese Vorgehensweise ist in Abbildung 6.9 dargestellt.

Die Werte, die diese Attribute annehmen können, richten sich immer nach den möglichen Andockstellen eines geometrischen Elements entsprechend der Definition der Eigenschaften dieser geometrischen Elemente. Die Zuweisung der Werte dieser Attribute erfolgt bei der Erstellung der jeweiligen Kanten durch eine Graphersetzungsregel. Dabei muss bei der

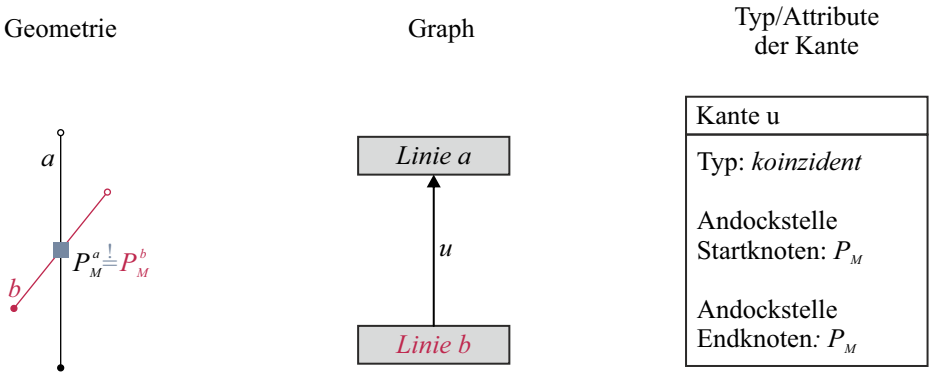


Abbildung 6.9: Die Information, welche Andockstellen bei Anfangs- und Endknoten einer Kante verwendet werden sollen, werden in den Werten der Attribute dieser Kante gespeichert.

Erstellung dieser Regeln sichergestellt werden, dass nur die für einen Knoten definierten Andockstellen verwendet werden. Ein Kante, deren Anfangsknoten einen Kreis repräsentiert, dürfte also im entsprechenden Attribut nicht den Wert für einen Startpunkt einer Linie enthalten. Die konkreten Bezeichnungen für die Attribute der Kanten und ihre möglichen Werte werden bei der Definition der entsprechenden Kanten festgelegt.

Dieses Vorgehen ist auch für die Implementierung der Interpretation des Graphen vorteilhaft, da die Andockstellen direkt aus den Attributen der Kante, die eine Zwangsbedingung repräsentiert, abgeleitet werden können. Beim Erstellen des entsprechenden Objekts in einem CAD-System müssen die Andockstellen daher nicht indirekt über die Knoten abgefragt werden, die die geometrischen Elemente repräsentieren, welche durch die Zwangsbedingung in Abhängigkeit zueinander gesetzt werden.

6.2.5 Temporäre Koordinaten

Da die Geometrie eines parametrischen Modells durch die Variation von Parametern verändert werden kann, können sich auch die Koordinaten der Objekte, aus denen sich das Modell zusammensetzt, verändern. Dies gilt insbesondere für die geometrischen Elemente, aus denen eine Skizze aufgebaut wird. Dieses Verhalten wurde bereits in Abschnitt 5.5.1 beschrieben.

Bei der Interpretation des Graphen werden die Informationen zur initialen Positionierung verwendet, um Objekte so zu platzieren, wie sie in einem manuellen Modellierungsprozess durch einen Nutzer platziert würden. Damit wird sichergestellt, dass das durch die Interpretation des Graphen erstellte evaluierte Modell dem von einem Anwender erwarteten Resultat entspricht. Weiterhin muss beachtet werden, dass die initiale Positionierung der Objekte das Verhalten des Geometric Constraint Solver (GCS) beeinflussen kann (siehe Abschnitt 3.2.4). Die durch die Erstellung parametrischer Zwangsbedingungen ausgelösten

Veränderungen der Geometrie hängen in vielen Fällen von der initialen Positionierung der Elemente durch den Nutzer ab.

Demnach muss die initiale Position eines geometrischen Elements in einer Skizze oder die eines Bauteils in einer Baugruppe in der graphbasierten Repräsentation enthalten sein. Hierzu werden den Knoten des Graphen, die diese Objekte repräsentieren, entsprechende Attribute zugewiesen. In den Werten dieser Attribute werden die temporären Koordinaten, die die initiale Position der Objekte im Modell definieren, gespeichert. Die Koordinaten beziehen sich immer auf das 2D-Koordinatensystem der Skizze, in der die geometrischen Elemente positioniert werden, bzw. auf das 3D-Koordinatensystem einer Baugruppe, in der ein Bauteil platziert wird. Diese Vorgabe muss bei der Interpretation des Graphen berücksichtigt werden.

Die Werte der Attribute, die die temporären Koordinaten speichern, werden im Zuge der Anwendung von Graphersetzungsregeln befüllt. In einer Graphersetzungsregel, durch deren Anwendung Knoten erstellt werden, die entsprechend ihres Typs Attribute für temporäre Koordinaten enthalten, muss daher die Zuweisung von Werten für diese Attribute berücksichtigt sein. Diese Werte können bei der Definition einer solchen Graphersetzungsregel entweder absolut festgelegt werden oder als Parameter der Graphersetzungsregeln variabel sein. Wird ein Wert über einen Parameter bestimmt, kann der Wert entweder von einem Nutzer abgefragt werden oder sich aus dem Wert eines Attributs des im Arbeitsgraph gefundenen Mustergraphen ergeben.

Die folgende Abbildung 6.10 zeigt links eine einfache graphbasierte Repräsentation einer Skizze, die zwei Punkte P_1 und P_2 enthält. Die beiden Punkte sind durch die Zwangsbedingung *horizontal* so voneinander abhängig, dass sie immer den gleichen Abstand von der x-Achse des Koordinatensystems der Skizze haben. Zusätzlich ist ihr Abstand durch eine dimensionale Zwangsbedingung festgelegt. Einer der Punkte ist weiterhin durch die Zwangsbedingung *fixiert* im Ursprung des Koordinatensystems der Skizze positioniert.

In diesem Beispiel sind für beide Punkte temporäre Koordinaten notwendig. Damit er durch die Zwangsbedingung *fixiert* am richtigen Ort festgehalten wird, muss der Punkt P_1 schon initial korrekt im Ursprung des Koordinatensystems platziert werden. Auch die finale Position des Punktes P_2 hängt von seiner initialen Positionierung ab. Je nachdem, ob seine temporäre x-Koordinate positiv oder negativ ist, wird er aufgrund des folgenden Anlegens der dimensionalen Zwangsbedingung entweder rechts oder links von P_1 final durch den Geometrical Constraint Solver während der Erstellung des evaluierten Modells positioniert. Die beiden möglichen Varianten der Skizze in Abhängigkeit des Vorzeichens der x-Koordinate von P_2 sind in Abbildung 6.10 auf der rechten Seite dargestellt.

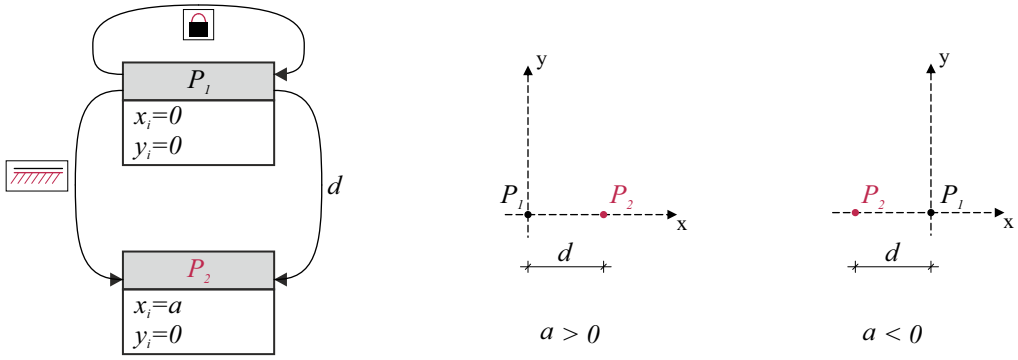


Abbildung 6.10: Darstellung zweier möglicher Varianten einer Skizze, die sich nur durch das Vorzeichen der im Graphen über die temporäre x-Koordinate festgelegten initiale Position des Punktes P_2 ergeben.

6.2.6 Nicht explizit im Graphen enthaltene Geometrie

Ein parametrisches Modell wird durch Modellierungsoperationen, die ein Nutzer ausführt, aufgebaut. Mittels des Graphersetzungssystems werden diese Modellierungsoperationen und dadurch auch indirekt das Ergebnis der Modellierung repräsentiert. Der Graph ist damit insgesamt eine implizite Repräsentation eines parametrischen Modells (siehe Abschnitt 3.1), in der aber auch explizite Informationen enthalten sind.

Je nachdem, ob eine Modellierungsoperation zur Erstellung von Geometrie[9] auf Skizzen- oder Bauteilebene durchgeführt wird, beschreibt diese Operation entweder explizit oder implizit, wie das Modell verändert wird. Wird beispielsweise eine Linie gezeichnet, ist das Ergebnis der Modellierungsoperation genau diese Linie in der Skizze eines Modells, also ein explizit beschriebenes geometrisches Objekt. Wird hingegen eine Extrusion ausgeführt, basiert diese zwar auf einer Skizze, die sich aus expliziten geometrischen Elementen zusammensetzt, es werden jedoch zusätzliche geometrische Elemente durch den Modellierkern der parametrischen CAD-Anwendung erstellt, mittels des das Modell interpretiert wird. Diese neuen geometrischen Elemente definieren sich aber nur implizit durch das Extrudieren der Skizze, da sie bei einer manuellen Modellierung nicht konkret durch einen Anwender erstellt werden. Der wesentliche Vorteil für den Anwender besteht hier in dieser automatischen Erstellung der Geometrie und der Möglichkeit den erstellen Körper nachträglich anzupassen, indem die Skizze oder die Länge der Extrusion verändert wird. Da die neu erstellte Geometrie auch direkt im Modell sichtbar ist, kann der Nutzer diese auch als Grundlage für weitere Modellierungsoperationen nutzen.

Dies spiegelt sich in der graphbasierten Repräsentation so wider, dass eine Graphersetzungsregel, mit der eine explizite Modellierungsoperation repräsentiert wird, auch direkt dem

[9] Damit sind nicht solche Modellierungsoperationen gemeint, die die Modellierung lediglich unterstützen (z. B. das Anlegen einer noch leeren Skizze).

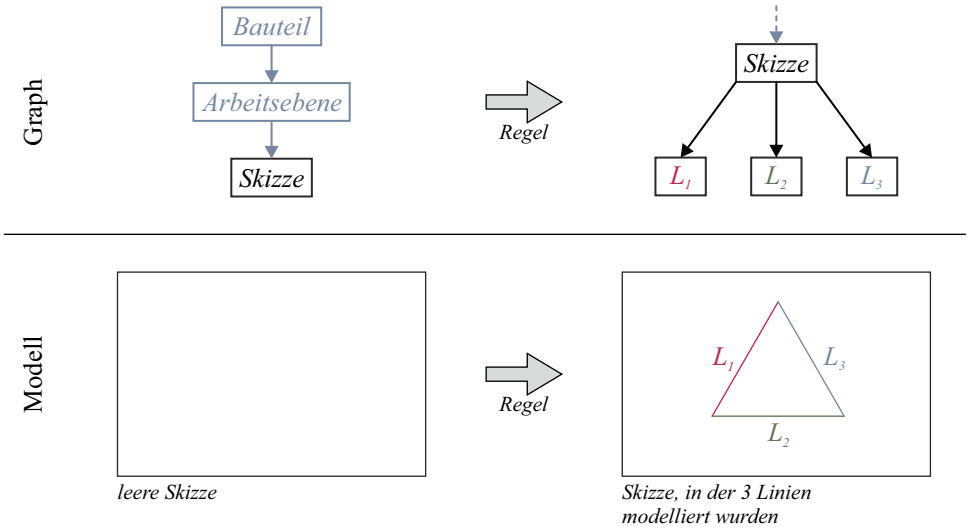


Abbildung 6.11: Durch die Anwendung einer Graphersetzungsregel werden drei Linien-Knoten zum Graphen hinzugefügt. Damit wird eine Modellierungsoperation repräsentiert, durch die drei Linien in einer Skizze erstellt werden. Diese Linien sind sowohl im Graphen als auch im Modell (als Teil der Skizze) konkret enthalten. Zu beachten ist, dass hier keinerlei parametrische Zwangsbedingungen enthalten sind, da diese für den in diesem Abschnitt beschriebenen Sachverhalt nicht relevant sind.

Graphen das entsprechende geometrische Element als Knoten hinzufügt. Dieser Knoten repräsentiert dann gleichzeitig die Modellierungsoperation und auch das Ergebnis der Operation in Form eines geometrischen Elements, das von weiteren Modellierungsoperationen (z. B. dem Hinzufügen einer parametrischen Zwangsbedingung) referenziert werden kann. Dies ist in Abbildung 6.11 dargestellt.

Bei impliziten Modellierungsoperationen, wie Sweeps und Extrusionen ist dies nicht der Fall. Hier wird im Graphen zwar repräsentiert, dass die Operation ausgeführt wird und auf welcher Basis (Skizzen und gegebenenfalls Pfade) dies geschieht, nicht jedoch das konkrete Ergebnis in Form der dadurch erstellten Geometrie (Abbildung 6.12). Würde der Graph auch die neu erstellte Geometrie konkret enthalten, so würde er letztlich nicht nur den Ablauf der Modellierung repräsentieren, sondern es müssten Graphersetzungsregeln konzipiert werden, die die umfangreichen Funktionalitäten eines parametrischen Modellierkern abbilden.

Die Folge wäre, dass nicht mehr der Ablauf der Modellierung aus Anwendersicht im Graphen repräsentiert ist, sondern letztlich der Graph zu einem vollständigen Datenmodell eines parametrischen CAD-Systems werden müsste. Da das in dieser Arbeit verfolgte Ziel aber darin besteht, den Ablauf einer parametrischen Modellierung zu repräsentieren, ist dies rein prinzipiell nicht sinnvoll. Weiterhin würde dadurch im Graphen ein zum genutzten CAD-System konkurrierendes Datenmodell entstehen. Dies würde die Interpretation des Graphen durch verschiedene CAD-System wesentlich erschweren, da sich die Datenmodelle und der Aufbau dieser Systeme so voneinander unterscheiden, dass es wie schon in Abschnitt 2.1.3 auf Seite 22 beschrieben, bisher nicht möglich war, ein neutrales Datenformat zum Austausch

parametrischer Modelle tatsächlich in der Praxis zu implementieren. Diese bereits bekannte Problematik ist letztlich ein wesentlicher Grund dafür, den Graphen so zu konzipieren, dass damit Graphersetzungsregeln zur Repräsentation parametrischer Modellierungsoperationen genutzt werden können und nicht als Datenmodell eines parametrischen CAD-Systems.

Nichtsdestoweniger ist es für die Erstellung komplexerer Modelle notwendig, Modellierungsoperationen vorzunehmen, die auf den nur implizit im Graphen repräsentierten Ergebnissen von vorherigen Modellierungsoperationen basieren. Daher muss es möglich sein, diese geometrischen Elemente mittels der graphbasierten Repräsentation zu referenzieren, auch wenn die Elemente nicht explizit im Graphen enthalten sind. Dies birgt den Vorteil, dass eine Modellierungsoperation auf Basis einer vorangegangenen Operation definiert werden kann, ohne dass das explizite (Zwischen-)Ergebnis dieser vorangegangenen Operation schon zum Zeitpunkt der Konzeption der jeweiligen Regel bekannt ist. Nur so können letztlich Graphersetzungsregeln konzipiert werden, die in verschiedenen Szenarien verwendbar sind.

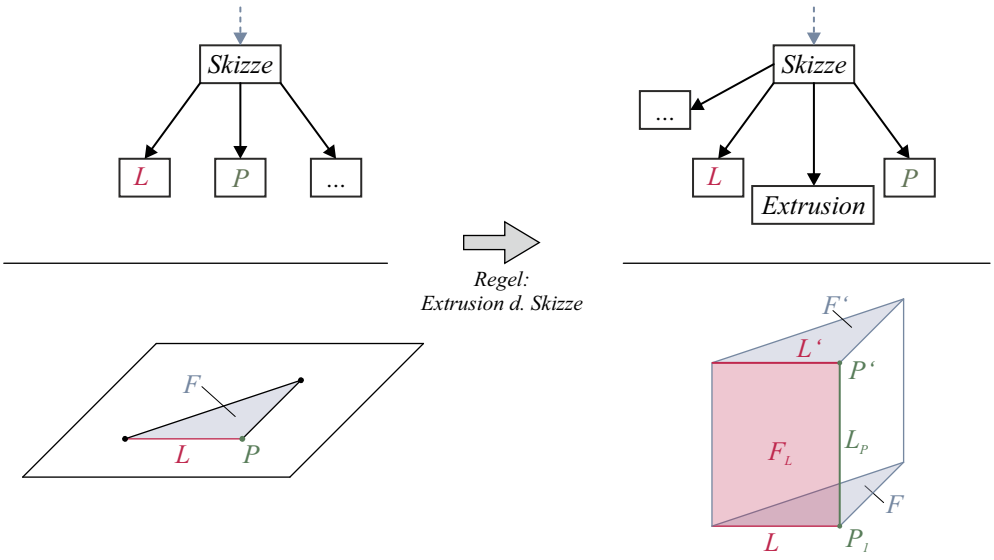


Abbildung 6.12: Durch die Anwendung einer Graphersetzungsregel wird ein Extrusions-Knoten zum Graph hinzugefügt. Damit repräsentiert der Graph neben den Modellierungsoperationen zum Erstellen der Skizze und der Linien auch die Extrusion dieser Skizze. Wird dieser Graph interpretiert, so entsteht ein Volumenmodell. Die dadurch im Modell neu erstellten Knoten, Kanten und Flächen (u.a. F_L, L', L_P, P' und F') sind im Graphen nicht explizit enthalten.

In diesem Kontext muss zuerst betrachtet werden, wie geometrische Objekte durch Sweeps und Extrusionen erstellt werden. Wie in Kapitel 3 beschrieben, entstehen im Zuge dieser Operationen Volumenmodelle auf Basis von Skizzen, die ein geschlossenes Profil definieren. Wie in Abbildung 6.12 dargestellt, findet dabei einerseits eine Erhöhung der Dimension der geometrischen Elemente in der Skizze statt: Punkte in der Skizze resultieren in einer Kante des Volumenkörpers und Linien werden zu Flächen, wobei die ursprünglichen Elemente weiterhin als Ecken bzw. Kanten des Volumenkörpers existieren. Weiterhin wird die Skizze im Raum verschoben, wodurch zusätzliche weitere Eckpunkte und Kanten des

Volumenkörpers entstehen. Genau diese neu erstellten Eckpunkte, Kanten und Flächen des Volumenkörpers werden nun im Graphen nicht repräsentiert, da ihre Erstellung Aufgabe des Modellierkerns des CAD-Systems ist, und sie aus Sicht des Nutzers zwar das Ergebnis einer Modellierungsoperation sind, aber nicht explizit durch den Nutzer erstellt werden. Wichtig ist hierbei allerdings, dass durch die bestehenden geometrischen Elemente der Skizze eine eindeutige Referenzierung aller neu erstellten geometrischen Elemente, die den Volumenkörper ausmachen, möglich ist. Soll also beispielsweise eine Arbeitsebene auf einer der Flächen des Volumenkörpers erstellt werden, kann diese Fläche auf Basis der geometrischen Elemente in der Skizze referenziert werden. Dies gilt genauso für Kanten, die beispielsweise in eine andere Skizze projiziert werden sollen. Betrachtet man den Volumenkörper als Bauteil[10], so ist die Referenzierung der Eckpunkte, Kanten und Flächen über die geometrischen Elemente in der Skizze auch für die Anwendung der Mating-Zwangsbedingungen auf Baugruppenebene relevant.

Abbildung 6.12 zeigt beispielhaft anhand des Punkts P und der Linie L, welche neuen geometrischen Elemente durch die Extrusion entstehen: Der Punkt P ist die Basis für die Linie L_P. Durch die Parallelverschiebung von P entsteht weiterhin P'. Analog ist L die Basis für die Fläche F_P. L' entsteht durch die Parallelverschiebung von L. Die weiteren Punkte und Linien sind in der Abbildung nicht näher gekennzeichneten, um eine übersichtliche Darstellung zu gewährleisten. Diese Punkte und Linien sind aber analog zu P und L die Basis zur Referenzierung der weiteren Linien, Punkte und Flächen, die durch die Extrusion entstehen. Weiterhin entsteht durch die Parallelverschiebung des geschlossenen Profils der Skizze aus der Fläche F die Fläche F', auf der eine weitere Arbeitsebene erstellt werden kann. Über diese Zusammenhänge ist nun eine Referenzierung der neu erstellten geometrischen Elemente F_L, L', L_P, P' und F' auf Basis von P, L und F möglich.

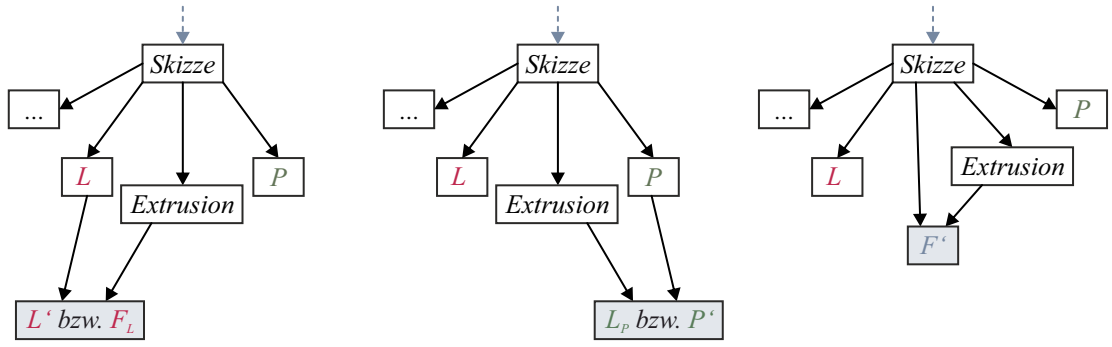


Abbildung 6.13: Auch wenn die neu erstellte Geometrie (siehe Abbildung 6.12) nicht explizit im Graphen repräsentiert wird, ist es möglich, diese Geometrie zu referenzieren. Dafür wird ein zusätzlicher Knoten benötigt. Die Kanten, die diesen Knoten mit dem Extrusions-Knoten und einem anderen bereits bestehenden Knoten verbinden, definieren, was dieser Knoten referenziert.

[10] Oder auch als Teil eines Bauteils.

In Abbildung 6.13 ist dargestellt, wie die Elemente eines durch Extrusion einer Skizze erstellten Volumenkörpers in der graphbasierten Repräsentation referenziert werden können.

Abschließend ist hier anzumerken, dass bei Volumenkörpern, die durch einen Sweep entlang eines nicht-linearen Pfades erstellt wurden, zwar grundsätzlich eine Referenzierung der so neu erstellten gekrümmten Kanten und Flächen möglich ist, diese hier aber nicht betrachtet werden. Dies bedingt sich dadurch, dass im hier vorgestellten Konzept keine Modellierungsoperationen definiert sind, die auf gekrümmten Kanten oder Flächen basieren.

6.3 Formale Definition des Graphen

Um ein parametrisches Modell mit Hilfe von Graphersetzungsregeln zu erstellen und zu detaillieren, muss die Zusammensetzung des Graphen, der zur Repräsentation des Modells genutzt wird, definiert werden. Dazu wird zuerst der Graph aus mathematischer Sicht definiert. Anschließend wird diese Definition durch ein Graph-Metamodell (siehe Abschnitt 4.2.3) vervollständigt, das die Typen von Knoten und Kanten sowie ihre Attribute konkret beschreibt. Wichtig hierbei ist, dass sich die folgende Definition auf die konkreten Graphen, die auf Basis des Metamodells instantiiert werden, bezieht.

Der Graph, der das prozedurale parametrische geometrische Modell repräsentiert, ist ein gerichteter attributierter getypter Multidigraph mit Schleifen:

$$G = (V, E, T^v, T^e, s, t, lb, ty^v, ty^e, att)$$

Formal wird dieser Graph wie folgt definiert:

- $V = V_A \cup V_P \cup V_S$ ist die nichtleere finite Menge der Knoten des Graphen.
  - Elemente in V_A repräsentieren Objekte auf Baugruppenebene.
  - Elemente in V_P repräsentieren Objekte auf Bauteilebene.
  - Elemente in V_S repräsentieren Objekte auf Skizzenebene.

  Diese Unterscheidung ist notwendig, um Aussagen treffen zu können, die sich auf alle Elemente einer Ebene beziehen.
- $E = E_A \cup E_P \cup E_S$ ist die nichtleere finite Menge der Kanten des Graphen.
  - Elemente in E_A repräsentieren Beziehungen zwischen Objekten auf Baugruppenebene.
  - Elemente in E_P repräsentieren Beziehungen zwischen Objekten auf Bauteilebene.
  - Elemente in E_S repräsentieren Beziehungen zwischen Objekten auf Skizzenebene.

Diese Unterscheidung ist notwendig, um Aussagen treffen zu können, die sich auf alle Elemente einer Ebene beziehen.

- Ein Element gehört immer genau zu einer Ebene:
  - $V_A \cap V_P = \emptyset, \qquad V_A \cap V_S = \emptyset, \qquad V_P \cap V_S = \emptyset$
  - $E_A \cap E_P = \emptyset, \qquad E_A \cap E_S = \emptyset, \qquad E_P \cap E_S = \emptyset$
- $s : E \to V$ ist eine Abbildung, die allen Kanten ihren Anfangsknoten zuordnet.
- $t : E \to V$ ist eine Abbildung, die allen Kanten ihren Endknoten zuordnet.
- Σ ist ein Alphabet, das die Bezeichnungen für die Knoten und Kanten des Graphen beinhaltet.
- $lb : E \cup V \to \Sigma$ ist eine Funktion, die allen Knoten und Kanten eine Bezeichnung aus Σ zuweist.
- $T^v = T^v_A \cup T^v_P \cup T^v_S$ ist die Menge der verschiedenen Typen der Knoten in V. T^v_A, T^v_P und T^v_S sind die Mengen der Knoten in V_A, V_P bzw. V_S. Damit ist definiert, welche Typen von Knoten in den verschiedenen Ebenen vorhanden sein können.
- $T^e = T^e_A \cup T^e_P \cup T^e_S$ ist die Menge der verschiedenen Typen der Knoten in E. T^e_A, T^e_P und T^e_S sind sind die Mengen der Knoten in E_A, E_P bzw. E_S. Damit ist definiert, welche Typen von Knoten in den verschiedenen Ebenen vorhanden sein können.
- $ty^v : V \to T^v$ ist eine Funktion zur Typisierung der Knoten. Dabei gilt, dass einem Knoten immer ein Typ entsprechend der Ebene, der er zugeordnet ist, zugewiesen wird:
  $ty^v(V_A) \cap ty^v(V_P) = \emptyset, \quad ty^v(V_A) \cap ty^v(V_S) = \emptyset, \quad ty^v(V_P) \cap ty^v(V_S) = \emptyset$
- $ty^e : E \to T^e$ ist eine Funktion zur Typisierung der Kanten. Dabei gilt, dass einer Kante immer ein Typ entsprechend der Ebene, der sie zugeordnet ist, zugewiesen wird:
  $ty^e(E_A) \cap ty^e(E_P) = \emptyset, \quad ty^e(E_A) \cap ty^e(E_S) = \emptyset, \quad ty^e(E_P) \cap ty^e(E_S) = \emptyset$
- At ist die Menge der Attribute, die den Knoten und den Kanten zugewiesen werden können.
- $att : E \cup V \to At$ ist eine Funktion, die einem Knoten oder einer Kante ein Attribut zuweist.

Die Attribute, die einem Knoten oder einer Kante zugewiesen werden, bedingen sich durch den Typ des Knotens bzw. der Kante. Im folgenden Abschnitt wird ein Metamodell für den Graphen definiert, in dem festgelegt ist, welche Typen verfügbar sind und welche Attribute sie aufweisen.

6.4 Graph-Metamodell

Im folgenden Abschnitt wird das Graph-Metamodell (siehe Abschnitt 4.2.3) des Graphersetzungssystems, das im Rahmen dieser Arbeit entwickelt wurde, beschrieben. Auf Basis der formalen Definition des Graphen als grundlegende Datenstruktur[11] wird über das Graph-Metamodell spezifiziert, wie genau die Knoten und Kanten des Graphen genutzt werden, um Informationen zu speichern und damit ein parametrisches Modell zu repräsentieren.

Im Unterschied zur OMG-Metamodell-Hierarchie wird in der ISO 11179 (2015) ein Metamodell vergleichsweise allgemein als *„Datenmodell, das ein oder mehrere andere Datenmodelle spezifiziert“* beschrieben[12]. Das übergeordnete Graph-Metamodell fasst diese Datenmodelle in einer formalen Definition zusammen, die als Klassenstruktur entsprechend dem Konzept der objektorientierten Programmierung aufgebaut ist. Dieses Vorgehen wurde dem Konzept einer objektorientierten Graphgrammatik von Helms (2013) nachempfunden, der hierzu die Funktionalitäten des Graphersetzungs-Tools GrGen.NET (Jakumeit *et al.*, 2021) einsetzt, das auch im Rahmen dieser Arbeit verwendet wird. Während in der objektorientierten Programmierung allerdings Objekte aus Klassen instanziiert werden, werden hier Knoten und Kanten aus den im Graph-Metamodell definierten Knotentypen und Kantentypen instanziiert.

Das Graph-Metamodell ist entsprechend der in Kapitel 5 aufgeführten Anforderungen definiert, um sicherzustellen, dass die graphbasierte Repräsentation die notwendigen Informationen aller Objekte beinhaltet, die notwendig sind, um ein parametrisches Modell zu repräsentieren. Auch die in diesem Kapitel beschriebenen Ebenen der Repräsentation werden – genauso wie die Vorüberlegungen zur Konzeption des Graphen – im Graph-Metamodell aufgegriffen. Damit ist das Graph-Metamodell letztlich die Synthese aller bisherigen Anforderungen und Vorüberlegungen.

Durch den objektorientierten hierarchischen Aufbau ist es außerdem möglich, das Graph-Metamodell flexibel zu erweitern, sodass zusätzliche parametrische Modellierungsoperationen abgebildet werden können. Bei Bedarf können dem Metamodell neue Knoten- und Kantentypen hinzugefügt und so in die hierarchische Struktur integriert werden, dass sie durch die Nutzung der Attributvererbung auf bereits bestehenden Festlegungen aufbauen. In diesem Kontext ist auch zu beachten, dass auf Basis des hier definierten Graph-Metamodells prinzipiell nicht nur ein einzelnes Graphersetzungssystem erstellt werden kann. Da sich ein Graphersetzungssystem aus dem Graph-Metamodell und einer Menge an Graphersetzungsregeln aufbaut, können beliebig viele verschiedene Graphersetzungssysteme, in denen dieses Graph-Metamodell zur Anwendung kommt, konzipiert werden. Mit solchen verschiedenen

[11] Nach der OMG-Metamodell-Hierarchie entspricht der Graph als grundlegende Datenstruktur der M2-Ebene (siehe auch Abbildung 6.1, S. 129). Das Graph-Metamodell spezifiziert diese Datenstruktur und entspricht demnach der M1-Ebene.

[12] Im engl. Original: *„data model that specifies one or more other data models“*.

Graphersetzungssystemen für unterschiedliche Modellierungsszenarien können verschiedene Arten von Modellen erstellt und verändert werden. Dies ist anhand der Beispiele in Kapitel 7 dargestellt. Werden Graphersetzungsregeln für ein neues Modellierungsszenario definiert, können durch die Erweiterbarkeit des Graph-Metamodells hierfür notwendige Knoten- und Kantentypen hinzugefügt werden. Sofern dabei der bestehende Teil des Graph-Metamodells nicht verändert wird, bleiben auch bereits definierte Graphersetzungsregeln weiter anwendbar. Dies ist insbesondere relevant, da sich das in der Modellerstellung enthaltene Wissen potenziell immer weiter entwickelt (Helms, 2013).

Im Wesentlichen definiert das Graph-Metamodell die Typen T^v der Knoten V und die Typen T^e der Kanten E des Graphen (entsprechend Abschnitt 6.3) im Detail. Dabei wird auch festgelegt, welche Attribute den Knoten und Kanten entsprechend ihrer Typen zugewiesen werden können. Wird ein Graph entsprechend des Graph-Metamodells erstellt, werden die Knoten und Kanten entsprechend der Typen instanziiert. Weiterhin werden Festlegungen zu den erlaubten und notwendigen Nachbarschaftsbeziehungen von Knoten und Kanten (Adjazenz, Inzidenz) getroffen, sodass die Topologie eines entsprechend des Graph-Metamodells aufgebauten Graphen von vorn herein eingeschränkt ist und tatsächlich ein parametrisches Modell repräsentiert. In der Umsetzung wurde dieses Konzept allerdings so verankert, dass die Erstellung eines Graphen, der diesen Festlegungen entspricht, nicht prinzipiell verboten ist. Das bedeutet, dass auf Basis des Graph-Metamodells ein Graph instantiiert werden kann, dessen Nachbarschaftsbeziehungen den definierten Vorgaben widersprechen. Es ist jedoch möglich, den Graphen auf diese Festlegungen hin zu validieren. So wird festgestellt, ob ein Graph entsprechend der Nachbarschaftsbeziehungen *valide* oder *invalide* ist. Dadurch kann beispielsweise nach Anwendung einer Graphersetzungsregel geprüft werden, ob diese Regel zu einem gültigen Resultat führt. Ist dies nicht der Fall kann die Anwendung der Regel rückgängig gemacht oder eine weitere Regel ausgeführt werden, die wieder einen validen Zustand herstellt. Der Grund für dieses Vorgehen besteht in der Möglichkeit, Gruppen von Regeln definieren zu können, die konsekutiv angewendet werden müssen, da als Zwischenergebnis ein nicht-valider Graph entsteht.

Das Graph-Metamodell bildet demzufolge auch die Grundlage zur Definition der Graphersetzungsregeln auf die im nächsten Abschnitt eingegangen wird. Da eine Graphersetzungsregel ausgeführt wird, um einen Graphen zu erstellen oder eine Veränderung an einem bestehenden Graphen vorzunehmen, müssen die im Graph-Metamodell definierten Vorgaben bei der Konzeption der Regeln immer berücksichtigt werden. Die Definition der Graphersetzungsregeln wird damit von Vornherein durch das Metamodell eingeschränkt, sodass keine Regeln definiert werden können, deren Anwendung in einem Graphen resultieren, der den Vorgaben des Graph-Metamodells widerspricht[13]. Damit dienen die Graphersetzungsregeln auch dazu,

[13] Dies bezieht sich lediglich auf die Vorgaben hinsichtlich der Typen und Attribute von Knoten und Kanten. Im Graph-Metamodell festgelegte Nachbarschaftsbeziehungen können durch eine Graphersetzungsregel verletzt werden, siehe voriger Absatz.

die im Graph-Metamodell getroffenen Definitionen zu Typen und Attributen umzusetzen. Im Zuge der Anwendung einer Graphersetzungsregel werden die Funktionen zur Typisierung (ty^v, ty^e) und Attributierung (att) ausgeführt, um Knoten oder Kanten, die durch die Regel neu erstellt werden, einen Typ und die entsprechenden Attribute zuzuweisen.

Das Metamodell des Graphen ist als Klassenstruktur aufgebaut, die die Vererbung von Attributen erlaubt. Dies schließt auch das Konzept der Mehrfachvererbung und die Verwendung abstrakter Klassen bzw. Typen mit ein. Neben den Attributen kann die hierarchische Struktur auch zur Vererbung von Definitionen zur validen Topologie des Graphen genutzt werden: Bei der Erstellung von Graphersetzungsregeln können die Vererbungsbeziehungen von Knoten und Kanten genutzt werden, um den Mustergraphen flexibler zu definieren[14].

6.4.1 Überblick über den Aufbau des Graph-Metamodells

Die Abbildung 6.14 gibt einen Überblick über die im Metamodell definierten Knoten- und Kantentypen und die durch den hierarchischen Aufbau definierten Vererbungen. Die verschiedenen Typen werden in den folgenden Abschnitten detailliert beschrieben und anhand von Beispielen erklärt. Dabei wird auch beschrieben, aus welchem Grund die Klassenstruktur so wie dargestellt konzipiert wurde. In den Abbildungen sind der Übersichtlichkeit halber hier noch keine Attribute dargestellt. Die Attribute werden in den folgenden Abschnitten im Zuge der detaillierten Beschreibung der einzelnen Typen aufgeführt.

Zusätzlich zum Überblick über die Hierarchie der Knoten- und Kantentypen in Abbildung 6.14 sind die verschiedenen Typen in den folgenden Tabellen 6.1, 6.2 und 6.3 inklusive einer kurzen Beschreibung aufgelistet.

Die Grundlage für den dargestellten Aufbau des Graph-Metamodells bilden die in Kapitel 5 definierten Elemente eines parametrischen Modellierungsprozesses und die im aktuellen Kapitel beschriebenen Vorüberlegungen. Das Graph-Metamodell ist damit das Resultat der Synthese dieser Randbedingungen und Überlegungen, die es ermöglicht ein parametrisches Modell als Graph zu repräsentieren. Der Zusammenhang zwischen den Elementen eines parametrischen Modellierungsprozesses und den entsprechenden Definitionen ist dabei meist klar erkennbar, teils aber auch nicht direkt ersichtlich. Daher sind auch diese Zusammenhänge zwischen einem parametrischen Modell und der graphbasierten Repräsentation durch detaillierten Beschreibungen in den folgenden Abschnitten erklärt.

[14] Ist in einem Mustergraph ein bestimmter Basistyp enthalten, werden bei der Suche nach einem isomorphen Subgraph auch die davon abgeleiteten Typen berücksichtigt. In einem Mustergraphen kann daher auch eine abstrakte Klasse enthalten sein.

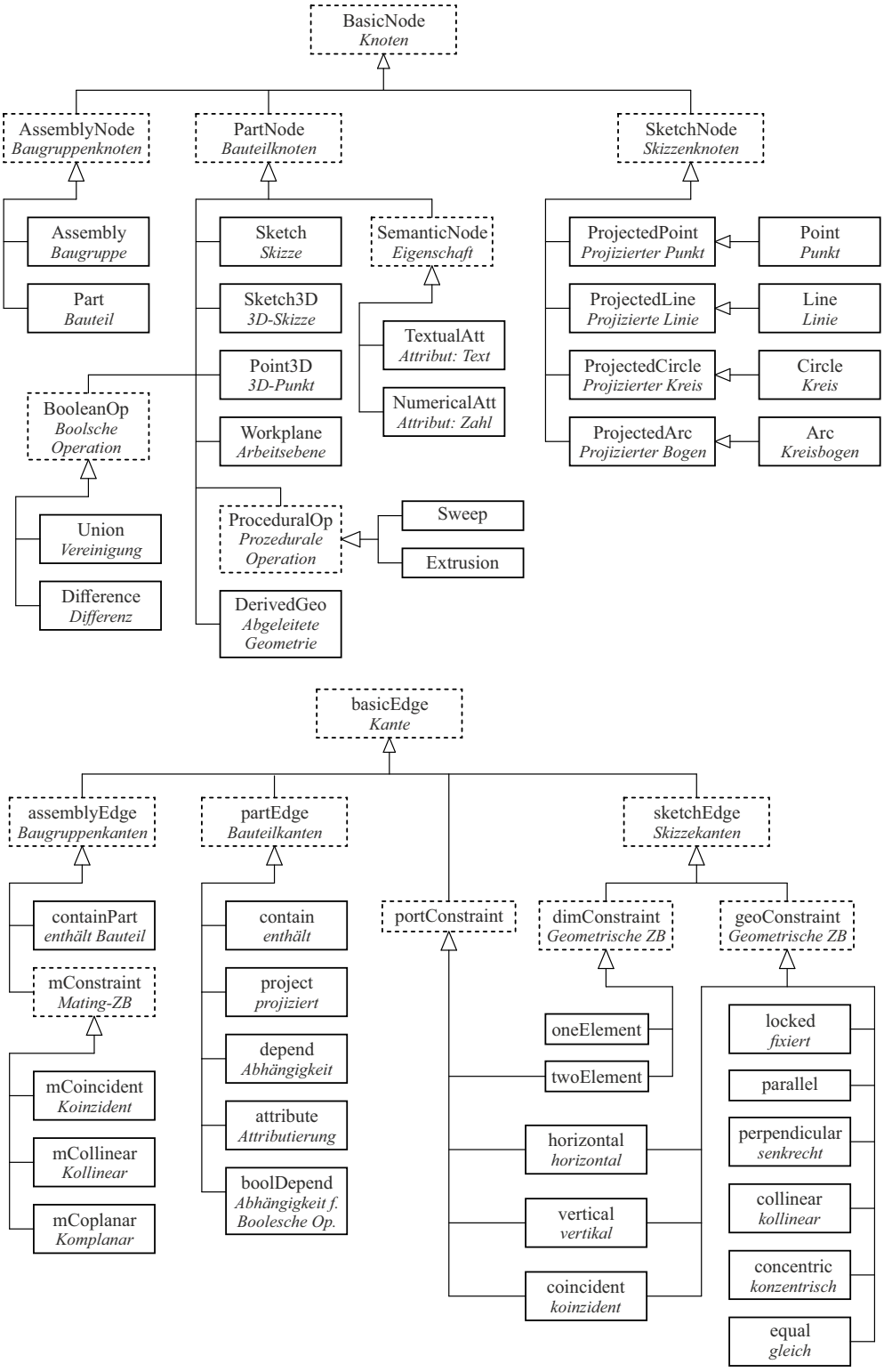


Abbildung 6.14: Klassendiagramm der Knoten- und Kantentypen, die durch das Graph-Metamodell definiert werden. Es sind nur die verschiedenen Typen und ihre Vererbungshierarchie, aber keine Attribute dargestellt. Abstrakte Kantentypen, sind durch gestrichelte Umrisse gekennzeichnet. *Die englischen Begriffe werden im Folgenden in allen Beispielen verwendet.*

Tabelle 6.1: Beschreibung der Knotentypen.

Knotentyp	**Beschreibung**
BasicNode	Abstrakter Knotentyp, von dem sich alle weiteren Knotentypen ableiten und über den Attribute, die für alle Knotentypen relevant sind, definiert werden können.
AssemblyNode	Abstrakter Knotentyp, über den festgelegt ist, dass auf Basis der Subtypen Knoten instantiiert werden, die Objekte auf Baugruppenenbene repräsentieren.
Assembly	Knoten dieses Typs repräsentieren eine Baugruppe.
Part	Knoten dieses Typs repräsentieren ein Bauteil.
PartNode	Abstrakter Knotentyp, über den festgelegt ist, dass auf Basis der Subtypen Knoten instantiiert werden, die Objekte auf Bauteilebene repräsentieren.
Sketch	Knoten dieses Typs repräsentieren eine Skizze in einem Bauteil.
Sketch3D	Knoten dieses Typs repräsentieren eine 3D-Skizze in einem Bauteil.
Point3D	Knoten dieses Typs repräsentieren einen 3D-Punkt, der Teil einer 3D-Skizze ist.
DerivedGeo	Über Knoten dieses Typs können geometrische Elemente, die nicht direkt im Graphen enthalten sind, referenziert werden. (siehe auch Abschnitt 6.2.6)
SweepingOp	Abstrakter Knotentyp von dem sich die Typen *Extrusion* und *Sweep* ableiten, mittels derer Extrusionen und Sweeps repräsentiert werden.
BooleanOp	Abstrakter Knotentyp von dem sich die Typen *Union* und *Difference* ableiten, mittels derer die Booleschen Operationen *Vereinigung* und *Differenz* repräsentiert werden.
SemanticNode	Textuelle und numerische Attribute zur Beschreibung der Semantik eines Körpers oder eines Bauteils können mittels Knoten der Typen *TextualAtt* und *NumericalAtt* repräsentiert werden. Dies ermöglicht auch die Modellierung von Attributen, die nicht dediziert im Graph-Metamodell definiert wurden.
SketchNode	Abstrakter Knotentyp über dessen Subtypen geometrischen Elemente auf Skizzenebene repräsentiert werden.
ProjectedPoint *Point*	Knoten dieser Typen repräsentieren einen Punkt oder einen projizierten Punkt in einer Skizze.
ProjectedLine *Line*	Knoten dieser Typen repräsentieren eine Linie oder eine projizierte Linie in einer Skizze.
ProjectedCircle *Circle*	Knoten dieser Typen repräsentieren einen Kreis oder einen projizierten Kreis in einer Skizze.
ProjectedArc *Arc*	Knoten dieser Typen repräsentieren einen Bogen oder einen projizierten Bogen in einer Skizze.

Tabelle 6.2: Beschreibung der Kantentypen auf Baugruppen- und Bauteilebene.

Kantentyp	**Beschreibung**
basicEdge	Abstrakter Kantentyp, von dem sich alle weiteren Kantentypen ableiten und über den Attribute, die für alle Kantentypen relevant sind definiert werden können.
assemblyEdge	Abstrakter Kantentyp, über den festgelegt ist, dass auf Basis der Subtypen Kanten instantiiert werden, die Zusammenhänge zwischen Objekten auf Baugruppenenbene repräsentieren.
containPart	Über Kanten dieses Typs wird ein Bauteil einer Baugruppe zugeordnet.
mConstraint	Abstrakter Kantentyp, dem die Kantentypen zur Repräsentation der Mating-Zwangsbedingungen untergeordnet sind.
mCoincident	Kanten dieses Typs definieren, dass zwei Punkte in verschiedenen Bauteilen innerhalb der Baugruppe koinzident sind.
mCollinear	Mittels Kanten dieses Typs wird festgelegt, dass zwei Linien in verschiedenen Bauteilen innerhalb der Baugruppe kollinear sind.
mCoplanar	Kanten dieses Typs definieren, dass zwei Flächen in verschiedenen Bauteilen innerhalb der Baugruppe komplanar sind.
partEdge	Abstrakter Kantentyp, über den festgelegt ist, dass auf Basis der Subtypen Kanten instantiiert werden, die Objekte auf Bauteilebene repräsentieren.
contain	Mittels Kanten dieses Typs werden einem Bauteil verschiedene Objekte (Arbeitsebenen, 3D-Skizzen etc.) bzw. einer Skizze geometrische Elemente zugeordnet.
project	Kanten dieses Typs dienen dazu festzulegen, auf welches geometrische Element sich ein projiziertes geometrisches Element bezieht.
depend	Die Kanten dieses Typs werden immer dann verwendet, wenn die Abhängigkeit eines Objekts von einem anderen Objekt definiert werden muss. Die verschiedenen Anwendungsfälle werden in den nächsten Abschnitten erklärt.
attribute	Kanten zur Festlegung, welchem Objekt eine Eigenschaft, die durch einen *SemanticNode*-Knoten definiert wird, zugeordnet ist.
boolDepend	Mittels Kanten dieses Typs wird festgelegt, welche Objekte durch eine Boolesche Operation kombiniert werden.

Tabelle 6.3: Beschreibung der Kantentypen auf Skizzenebene.

Kantentyp	**Beschreibung**
sketchEdge	Abstrakter Kantentyp für die Subtypen zur Repräsentation parametrischer Zwangsbedingungen auf Skizzenebene repräsentiert werden.
portConstraint	Abstrakter Kantentyp, von dem alle Kantentypen erben, bei denen sowohl für den Anfangs- als auch für den Endknoten eine Andockstelle definiert werden muss.
dimConstraint	Abstrakter Kantentyp, von dem alle dimensionalen Zwangsbedingungen ein Attribut zur Speicherung der Länge erben.
oneElement	Knoten dieses Typs repräsentieren eine dimensionale Zwangsbedingung, die sich nur auf ein geometrisches Element bezieht.
twoElement	Knoten dieses Typs repräsentieren eine dimensionale Zwangsbedingung, die sich nur auf zwei geometrische Elemente bezieht, um einen Abstand zwischen diesen Elementen zu definieren.
geoConstraint	Abstrakter Kantentyp, dem alle geometrischen Zwangsbedingungen untergeordnet sind.
locked	Kanten dieses Typs repräsentieren die Zwangsbedingung *fixiert*.
parallel	Kanten dieses Typs repräsentieren die Zwangsbedingung *parallel*.
perpendicular	Kanten dieses Typs repräsentieren die Zwangsbedingung *senkrecht*.
collinear	Kanten dieses Typs repräsentieren die Zwangsbedingung *kollinear*.
concentric	Kanten dieses Typs repräsentieren die Zwangsbedingung *konzentrisch* .
horizontal	Kanten dieses Typs repräsentieren die Zwangsbedingung *horizontal*.
vertical	Kanten dieses Typs repräsentieren die Zwangsbedingung *vertikal*.
equal	Kanten dieses Typs repräsentieren die Zwangsbedingung *gleich*.
coincident	Kanten dieses Typs repräsentieren die Zwangsbedingung *koinzident*.

6.4.2 Implementierung des Graph-Metamodells

Um ein Graphersetzungssystem sinnvoll verwenden zu können, bedarf es einer geeigneten Softwareumgebung, die eine automatisierte Ausführung von Graphtransformationen entsprechend den Graphersetzungsregeln erlaubt[15]. Im Rahmen dieser Arbeit wird das *Software Development Tool* GrGen.NET (Graph Rewrite GENerator) eingesetzt, welches am Karlsruher Institut für Technologie entwickelt wird (siehe Abschnitt 4.2.6). GrGen.NET bietet eine Programmierumgebung, in der ein Graphersetzungssystem bestehend aus Graph-Metamodell und Graphersetzungsregeln implementiert werden kann. Graphersetzungsregeln und Graph-Metamodell müssen dabei für das gewünschte Anwendungsszenario

[15]Welche Regeln ausgeführt werden sollen, entscheidet dabei der Nutzer, die Software sorgt anschließend dafür, dass die entsprechende Graphtransformation tatsächlich durchgeführt und der Graph so verändert wird.

definiert werden, während die tatsächliche Ausführung der Graphersetzungsoperationen, also das Finden eines Mustergraphen und das anschließende Einfügen des Ersetzungsgraphen durch GrGen.NET vorgenommen wird. Zur Nachvollziehbarkeit der Implementierung in GrGen.NET ist in den folgenden Abschnitten zusätzlich zur UML- und textbasierten Definition des Graph-Metamodell auch die entsprechende Umsetzung in der Syntax der durch GrGen.NET definierten *Graph Model Language* enthalten. Hinsichtlich der Definition, Syntax und weiterführenden Beschreibung dieser Sprache wird der interessierte Leser auf die umfangreiche Dokumentation[16] von GrGen.NET verwiesen (Jakumeit *et al.*, 2021).

Übergeordnete Knoten- und Kantentypen

Entsprechend der in Abschnitt 6.3 aufgeführten formalen Definition des Graphen muss jeder Knoten und jede Kante einer der Ebenen der graphbasierten Repräsentation zugeordnet sein. Diese Zuordnung wird im Graph-Metamodell über die in den Abbildungen 6.15 und 6.16 dargestellten abstrakten Knoten- bzw. Kantentypen sichergestellt. Knoten bzw. Kanten, die aus den Subtypen dieser abstrakten Typen instantiiert werden, sind der entsprechenden Ebene der graphbasierten Repräsentation zugeordnet.

Diese dargestellten abstrakten Typen sind wiederum Subtypen der beiden Basistypen *BasicNode* und *basicEdge* von denen sich alle Knoten- bzw. Kantentypen ableiten. Diese Basistypen leiten sich letztlich vom Graphen als Menge von Knoten und Kanten, der die grundlegende Datenstruktur auf M2-Ebene[17] definiert, ab.

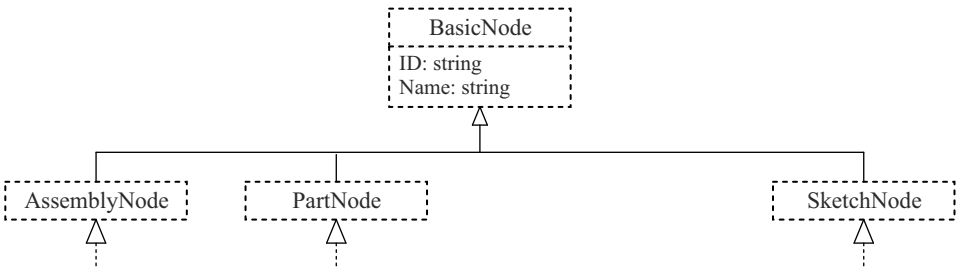


Abbildung 6.15: Durch diese abstrakten Knotentypen wird festgelegt, welcher Ebene der graphbasierten Repräsentation ein Knoten entsprechend des Typs, mittels dem er instantiiert wurde, zugeordnet ist.

Im folgenden Beispiel ist die Definition dieser Typen in GrGen.NET dargestellt. Das in den Abbildungen dargestellte Attribut *ID* als Label zur eindeutigen Identifikation eines im Graphen instantiierten Objekts[18], ist in dieser Definition nicht enthalten, da GrGen.NET generell allen Objekten im Graphen einen eindeutigen und dauerhaften Namen (engl. *persistent name*) zuweist, ohne dass dies explizit in der Definition des Graph-Metamodells

[16] Unter http://www.info.uni-karlsruhe.de/software/grgen/ abrufbar.
[17] Entsprechend der OMG-Metamodell-Hierarchie.
[18] Also eines Knotens oder einer Kante.

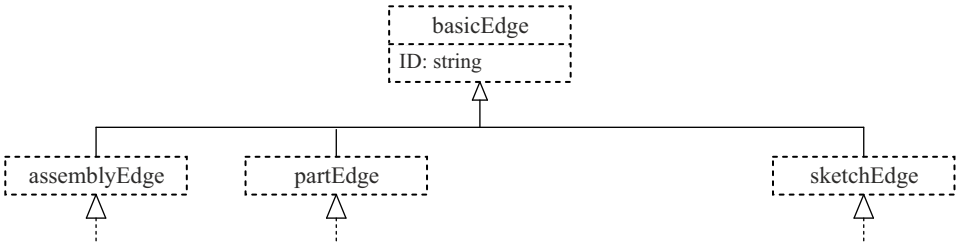


Abbildung 6.16: Durch diese abstrakten Kantentypen wird festgelegt, welcher Ebene der graph-basierten Repräsentation eine Kante entsprechend des Typs, mittels dem sie instantiiert wurde, zugeordnet ist.

enthalten sein muss. Insofern ist hier nur für den Knotentyp *BasicNode* ein Attribut *name* deklariert.

```
1  abstract node class BasicNode{
2      name:string;
3  }
4  abstract node class AssemblyNode extends BasicNode;
5  abstract node class PartNode extends BasicNode;
6  abstract node class SketchNode extends BasicNode;
7
8  abstract edge class basicEdge;
9  abstract edge class assemblyEdge extends basicEdge;
10 abstract edge class partEdge extends basicEdge;
11 abstract edge class sketchEdge extends basicEdge;
```

Listing 6.1: Definition der in den Abbildungen 6.15 und 6.16 dargestellten abstrakten Knoten- bzw. Kantentypen in GrGen.NET.

Knoten- und Kantentypen auf Baugruppenebene

In Abbildung 6.17 und in Listing 6.2 sind die Knoten- und Kantentypen auf Baugruppenebene sowie ihre Attribute definiert. Über Knoten der Typen *Assembly* und *Part* werden im Graphen die Bauteile, aus denen sich eine Baugruppe zusammensetzt, repräsentiert. Da in dieser Arbeit keine Verschachtelung von Baugruppen (siehe auch Abschnitt 5.3) vorgesehen ist, darf maximal ein *Assembly*-Knoten[19] in einem Graphen vorhanden sein. Falls in einem Graphen kein *Assembly*-Knoten existiert, beschreibt der Graph nur ein einzelnes Bauteil, das keiner Baugruppe zugeordnet ist. In diesem Fall darf maximal ein *Part*-Knoten vorhanden sein.

Mittels Kanten vom Typ *containPart* wird ein Bauteil einer Baugruppe zugeordnet. Über die Attribute dieser Kante wird die initiale Position des Bauteils innerhalb der Baugruppe bestimmt, wobei hier standardmäßig die Werte für eine Positionierung im Ursprung des

[19] Damit ist ein Knoten vom Typ *Assembly* gemeint. Im Folgenden wird weitgehend diese verkürzte Schreibweise verwendet.

Koordinatensystems[20] eingetragen werden. Die Kante muss von der Baugruppe auf das Bauteil zeigen[21]. Daher kann für Kanten dieses Typs festgelegt werden, dass sie immer von einem *Assembly*-Knoten auf einen *Part*-Knoten verweisen müssen. Da ein Bauteil nur einer Baugruppe zugeordnet werden darf, wird hier weiterhin festgelegt, dass maximal eine *containPart*-Kante inzident zu einem *Part*-Knoten sein darf. In Listing 6.2 wird dies über die Anweisung *connect* definiert. Hier wird weiterhin festgelegt, dass zu jedem *Assembly*-Knoten mindestens eine *containPart*-Kante inzident sein muss, da eine Baugruppe ohne Bauteile keine Geometrie enthalten würde und damit kein sinnvolles Modell darstellt.

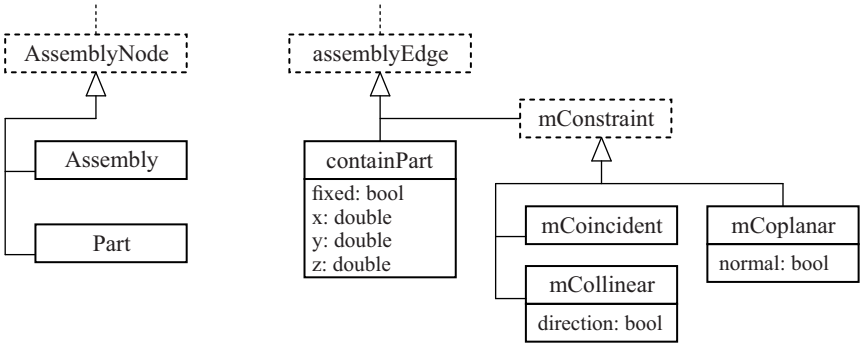


Abbildung 6.17: Knoten- (links) und Kantentypen (rechts) auf Baugruppenebene. Hier sind zusätzliche die Attribute, die für diese Typen definiert wurden inklusive des Datentyps aufgeführt.

Über das Attribut *fixed* kann festgelegt werden, ob ein Bauteil fest an seiner initialen Position verankert wird oder durch *Mating*-Zwangsbedingungen an einem andern Bauteil ausgerichtet werden kann, sodass sich seine Position dadurch verändert. Die *Mating*-Zwangsbedingungen werden mittels der *mConstraint*-Kanten definiert, die immer zwei Knoten zur Repräsentation konkreter oder abgeleiteter[22] geometrischer Elemente verbinden. Da diese geometrischen Elemente immer eindeutig einem Bauteil zugeordnet sind, ist damit auch immer implizit definiert, welche Bauteile aneinander ausgerichtet werden. Die Kante zeigt dabei auf ein geometrisches Element des Bauteils, das ausgerichtet werden soll und dadurch seine Position innerhalb der Baugruppe verändert[23].

Die *mConstraint*-Kanten sind hinsichtlich der Knoten, die sie verbinden dürfen, wie folgt eingeschränkt:

[20] $x = 0, y = 0, z = 0$

[21] Tatsächlich muss die Kante von einem Knoten, der eine Baugruppe repräsentiert, auf einen Knoten, der ein Bauteil repräsentiert, zeigen. Im Folgenden wird weitgehend die hier im Text verwendete verkürzte Formulierung benutzt, sofern sich aus dem Kontext ergibt, dass tatsächlich ein Knoten oder eine Kante gemeint ist.

[22] Siehe auch Abschnitt 6.2.6.

[23] Dies ist allerdings nicht zwingend der Fall. Falls eines der Bauteile als *fixiert* definiert ist, wird immer die Position des anderen Bauteils verändert.

- *mCoincident*: Anfangs- und Endknoten dieser Kanten sind jeweils entweder *ProjectedPoint*- *DerivedGeo*-Knoten oder Subtypen dieser Knotentypen[24].

- *mCollinear*: Anfangs- und Endknoten dieser Kanten sind jeweils entweder *ProjectedLine*-, *DerivedGeo*-Knoten oder Subtypen dieser Knotentypen.

- *mCoplanar*: Sowohl Anfangs- als auch Endknoten dieser Kanten sind *DerivedGeo*-Knoten.

```
1  // Nodes
2  abstract node class AssemblyNode extends BasicNode;
3  node class Assembly extends AssemblyNode;
4  node class Part extends AssemblyNode;
5
6  // Edges
7  abstract directed edge class assemblyEdge extends basicEdge;
8  directed edge class containPart extends assemblyEdge
9      connect Assembly[1:*] --> Part[0:1]
10 {
11     fixed: boolean = false;
12     x:double = 0;
13     y:double = 0;
14     z:double = 0;
15 }
16 abstract directed edge class mConstraint extends assemblyEdge
17     connect DerivedGeo[*] ?--? DerivedGeo[*];
18 directed edge class mCoincident extends mConstraint
19     connect ProjectedPoint[*] ?--? ProjectedPoint[*], copy extends;
20 directed edge class mCollinear extends mConstraint
21     connect ProjectedLine[*] ?--? ProjectedLine[*], copy extends
22     {direction:boolean = false;}
23 directed edge class mCoplanar extends mConstraint
24     connect copy extends
25 {normal:boolean = false;}
```

Listing 6.2: Definition der Knoten- bzw. Kantentypen auf Baugruppenebene in GrGen.NET.

Knoten- und Kantentypen auf Bauteilebene

In Abbildung 6.18 sind die Knoten- und Kantentypen auf Bauteilebene sowie ihre Attribute definiert. Die entsprechende Definition in GrGen.NET ist aufgrund des Umfangs in Anhang A aufgeführt. Anhang A enthält die vollständige Definition des Graph-Metamodels in der *Graph Model Language* inklusive der bereits in diesem Kapitel separat aufgeführten Teile.

[24] Da der Knotentyp *Point* ein Subtyp des Typs *ProjectedPoint* ist, reicht es aus diesen Typen zu benennen. Über die hierarchische Struktur der Knoten- und Kantentypen wird auch vererbt, welche Knotentypen zu einer Kante inzident sein dürfen.

Die Knoten auf dieser Ebene entsprechen dabei den Modellierungsoperationen, die bereits in Kapitel 5 beschrieben wurden. Abbildung 6.19 zeigt die im Folgenden beschriebenen Zusammenhänge beispielhaft.

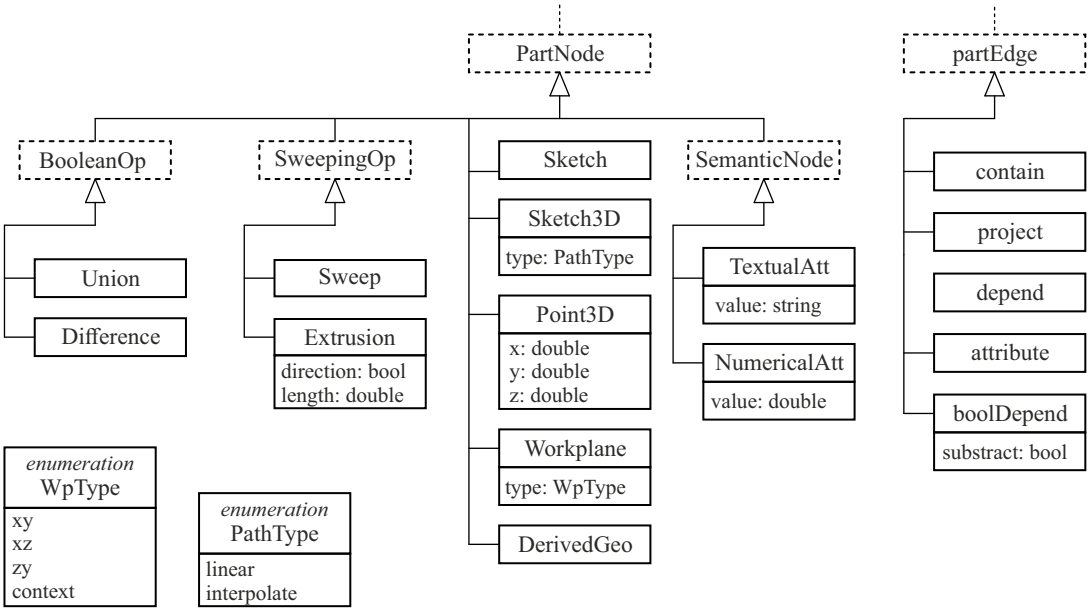


Abbildung 6.18: Knoten- (links) und Kantentypen (rechts) auf Bauteilebene. Hier sind zusätzliche die Attribute, die für diese Typen definiert wurden, inklusive des Datentyps aufgeführt.

Die grundlegendsten Operationen zur Erstellung der Geometrie eines Bauteils werden über die Knotentypen *Workplane* und *Sketch* zur Repräsentation von Arbeitsebenen bzw. Skizzen im Graphen abgebildet. Knoten dieser beiden Typen werden einem Bauteil (also einem *Part*-Knoten) über Kanten des Typs *contain* zugeordnet. *Sketch*-Knoten sind hier indirekt über einen *Workplane*-Knoten mit einem *Part*-Knoten verbunden: Vom *Part*-Knoten zeigt eine *contain*-Kante auf den *Workplane*-Knoten von dem eine weitere *contain*-Kante auf den *Sketch*-Knoten zeigt. Auch die Knoten zur Repräsentation von geometrischen Objekten in einer Skizze sind dieser Skizze über *contain*-Kanten zugeordnet.

Wenn ein *Workplane*-Knoten eine Arbeitsebene repräsentieren soll, die einer der Ursprungsebenen des Bauteil-Koordinatensystems entspricht, muss der Wert des Attributs *type* dieses Knotens *xy*, *xz* oder *zy* (aus der Aufzählung *WpType*) sein. Soll die Arbeitsebene von einer bestimmten 3D-Skizze abhängig sein (orthogonal zum Verlauf des durch diese 3D-Skizze repräsentierten Pfades an einem bestimmten 3D-Punkt auf diesem Pfad), so muss das Attribut *type* des entsprechenden *Workplane*-Knotens mit dem Wert *context* belegt sein. Die Abhängigkeit von der 3D-Skizze (und einem 3D-Punkt in dieser Skizze) wird über eine *depend*-Kante hergestellt. Diese Kante verbindet den entsprechenden *Workplane*-Knoten mit dem *Point3D*-Knoten, der diesen 3D-Punkt repräsentiert.

Weiterhin kann eine Arbeitsebene auf der Fläche eines Körpers, der durch eine Sweeping Modellierungsoperation erzeugt wurde, erstellt werden. Dies wird über eine Kante des Typs

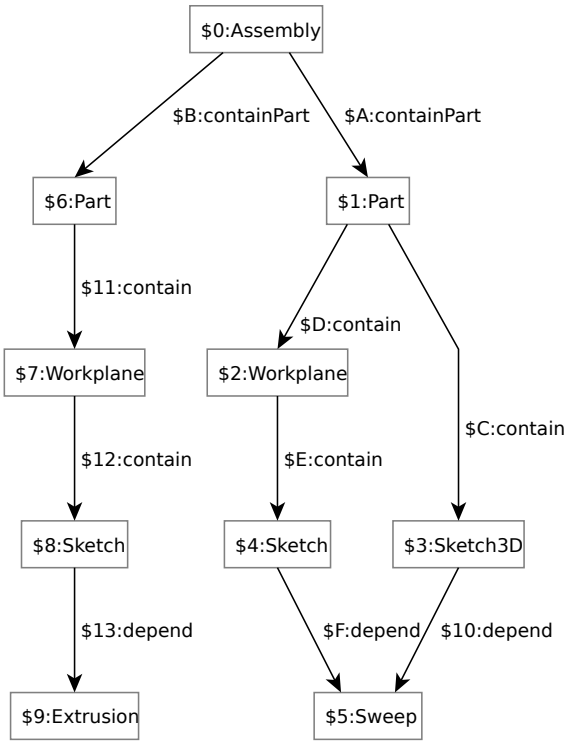


Abbildung 6.19: Der hier dargestellte Graph repräsentiert eine Baugruppe, die aus zwei Bauteilen zusammengesetzt ist. Part $6 enthält einen Körper der auf Basis der Skizze $8 extrudiert wird (repräsentiert durch den *Extrusion*-Knoten $9), während Part $1 zusätzlich eine 3D-Skizze $3 enthält, die zusammen mit Skizze $4 für eine Sweeping-Operation genutzt wird. *Die Attribute sowie Knoten und Kanten durch die die 3D-SKizze und die beiden Skizzen definiert werden, sind hier nicht enthalten.*

depend definiert. Diese Kanten zeigt von einem *DerivedGeo*-Knoten, auf den *Workplane*-Knoten. Der *DerivedGeo*-Knoten ist dabei von einem *SweepingOp*-Knoten und einem Knoten abhängig, der definiert, welche Fläche, die durch die Sweeping Operation erstellt wurde, gemeint ist. Dadurch entscheidet sich, ob die Arbeitsebene auf einer der Seitenflächen (die auf einer der Linien der extrudierten/gesweepten Skizze basiert) erstellt werden soll, oder auf der durch die Sweeping Operation im Raum verschobenen Grundfläche der prozeduralen Operation. Die verschiedenen Möglichkeiten sind in Abbildung 6.20 anhand eines extrudierten Dreiecks dargestellt.

Auch ein *Sketch3D*-Knoten zur Repräsentation einer 3D-Skizze ist einem Bauteil über eine *contain*-Kante zugeordnet. Da eine 3D-Skizze hier nur genutzt wird, um einen Pfad für Sweeping-Operationen zu definieren, enthält diese Skizze nur 3D-Punkte. Diese 3D-Punkte werden über Koordinaten (die Attribute x, y und z des *Point3D*-Knotens) innerhalb eines Bauteils platziert. Die entsprechenden *Point3D*-Knoten werden den Knoten des Typs *Sketch3D* (zur Repräsentation der 3D-Skizze) wiederum über *contain*-Kanten zugeordnet. Die 3D-Punkte definieren entweder einen Pfad aus linearen Segmenten oder einen Spline, der durch diese Punkte verläuft (dies wird über das *type*-Attribute des *Sketch3D*-Knotens

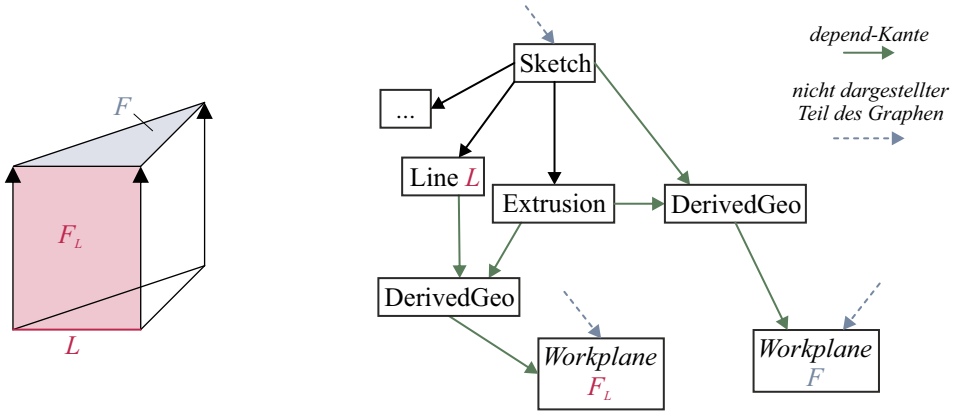


Abbildung 6.20: Eine Arbeitsebene kann auf einer Fläche eines Körpers, der durch eine Extrusion erzeugt wurde erstellt werden, indem die gewünschte Fläche mittels eines *DerivedGeo*-Knotens referenziert wird. Zeigt eine *depend*-Kante vom *Line*-Knoten L auf den *DerivedGeo*-Knoten, wird die Arbeitsebene auf der Fläche F_L erstellt. Soll die Fläche F referenziert werden, muss der *DerivedGeo*-Knoten mit dem *Sketch*-Knoten, der die ursprüngliche Skizze repräsentiert, verbunden sein.

festgelegt: *linear* bzw. *interpolate*). Die Reihenfolge, in denen der Pfad durch diese Punkte verläuft, wird über Kanten des Typs *depend* definiert, die immer zwei *Point3D*-Knoten verbinden und so eine Kette von *Point3D*-Knoten erzeugen. Dies ist beispielhaft in Abbildung 6.21 dargestellt.

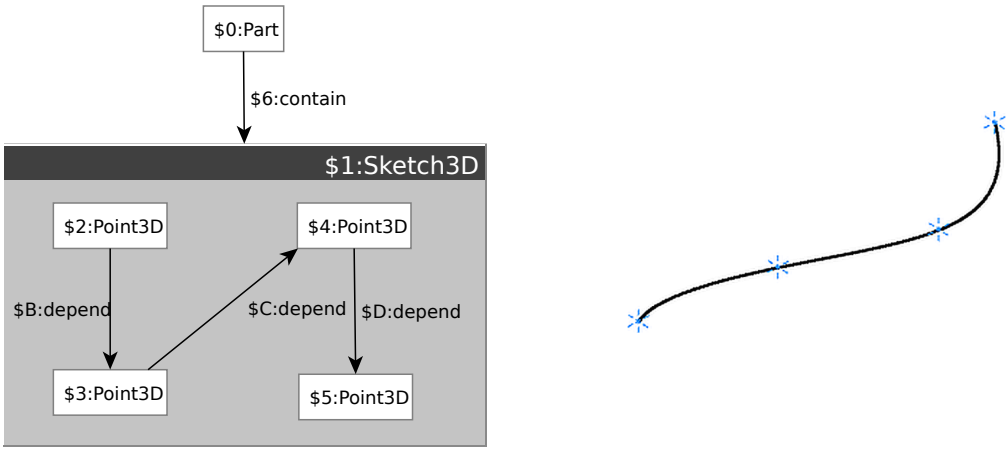


Abbildung 6.21: Mittels des Graphen auf der linken Seite wird ein Bauteil repräsentiert, dass eine 3D-Skizze enthält. Die 3D-Punkte dieser Skizze definieren den Verlauf des durch die 3D-Skizze definierten Pfads (rechts). Die *contain*-Kanten, die vom *Sketch3D*-Knoten auf die *Point3D*-Knoten zeigen, sind hier bei nicht visualisiert. Stattdessen sind diese Knoten im *Sketch3D*-Knoten gruppiert, um die visuelle Darstellung zu verbessern (siehe auch Abschnitt 4.1.2). Grundsätzlich sind diese Kanten aber im im Graphen enthalten.

Um auf Basis einer Skizze einen dreidimensionalen Körper zu erzeugen, sind prozedurale Modellierungsoperationen notwendig. Eine Extrusion wird rein auf Basis einer Skizze erstellt. Dies wird über eine *depend*-Kante, die vom *Sketch*-Knoten auf den *Extrusion*-Knoten zeigt, repräsentiert. Über die Attribute des *Extrusion*-Knotens wird die Länge der

Extrusion definiert und festgelegt, ob die Richtung des Extrusion dem Normalenvektor der Arbeitsebene der Skizze entspricht (*direction = true*) oder in die entgegengesetzte Richtung ausgeführt wird (*direction = false*). Auch zur Repräsentation einer Sweeping-Operation wird eine *depend*-Kante genutzt, die vom *Sketch*-Knoten auf den *Sweep*-Knoten zeigt. Damit wird wie bei der Extrusion die Grundfläche der Operation definiert. Da eine Sweeping-Operation zusätzlich zur Grundfläche einen Pfad benötigt, muss dieser über eine weitere *depend*-Kante referenziert werden, die von einem *Sketch3D*-Knoten, der den Pfad repräsentiert, auf den *Sweep*-Knoten zeigt.

Auf Basis der so erstellten dreidimensionalen Körper können Boolesche Operationen durchgeführt werden. Sollen zwei (oder mehr) Körper vereinigt werden, wird dies über einen *Union*-Knoten repräsentiert. Dazu werden Kanten des Typs *boolDepend* genutzt, die von den entsprechenden *SweepingOp*-Knoten auf den *Union*-Knoten zeigen. Für das Attribut *substract* der *boolDepend*-Kante wird in diesem Fall der Wert immer auf *false* gesetzt. Mittels Knoten des Typs *Difference* wird repräsentiert, dass ein Körper von einem anderen Körper abgezogen wird. Auch hier werden *boolDepend*-Kanten genutzt, um dem *Difference*-Knoten die durch *SweepingOp*-Knoten erstellten dreidimensionalen Körper zuzuordnen. Je nachdem, ob ein Körper abgezogen werden soll oder nicht, wird das *substract*-Attribut der *boolDepend*-Kante auf *true* oder *false* festgelegt.

Soll einem dreidimensionalen Körper, einem Bauteil oder einer kompletten Baugruppe eine Semantik (ein oder mehrere Eigenschaften[25]) zugeordnet werden, kann dies über *SemanticNode*-Knoten erfolgen. Die Repräsentation von Eigenschaften über einen eigenen Knotentyp hat den wesentlichen Vorteil, dass den Objekten beliebige Eigenschaften mit beliebigen Werten zugewiesen werden können. Würden die Eigenschaften als Attribute der jeweiligen Knotentypen im Graph-Metamodell definiert werden, müssten alle möglichen Attribute bereits während der Definition des Metamodells festgelegt werden. Durch die Anwendung einer Graphersetzunsgregel könnte in diesem Szenario nur noch der Wert aber nicht mehr die Bezeichnung einer Eigenschaft angepasst werden, was die Flexibilität deutlich einschränkt. Je nachdem, ob in einer Eigenschaft ein Text oder eine Zahl gespeichert werden soll, wird entweder ein *TextualAtt*- oder ein *NumericalAtt*-Knoten verwendet. Der Teil eines Modells, der mit einer Eigenschaft versehen werden soll, also ein *Assembly*-, ein *Part*-, ein *SweepingOp*- oder ein *BooleanOp*-Knoten, wird mittels einer Kante vom Typ *attribute* mit dem entsprechenden *TextualAtt*- bzw. *NumericalAtt*-Knoten verbunden. Der Wert der Eigenschaft wird über das *value*-Attribut des *SemanticNode*-Knotens definiert.

Weiterhin können auf Baugruppenebene Knoten des Typs *DerivedGeo* instantiiert werden. Mit diesen Knoten werden keine konkreten Modellierungsoperationen repräsentiert. Vielmehr dienen diese Knoten dazu, bestimmte Teile eines durch prozedurale Modellierungs-

[25] Es ist hervorzuheben, dass hier die *Eigenschaften* oder *Merkmale* eines mittels des Graphen repräsentierten Objekts gemeint ist und nicht die *Attribute* eines Knotens oder einer Kante.

operationen erstellten Körpers zu referenzieren, die nicht explizit im Graphen enthalten sind (siehe Abschnitt 6.2.6). Die Referenzierung wird dabei über *depend*-Kanten hergestellt. Zeigen beispielsweise eine *depend*-Kante von einem *Line*-Knoten und eine weitere *depend*-Kante von einem *Extrusion*-Knoten auf einen *DerivedGeo*-Knoten, so wird durch diesen Knoten die Linie referenziert, die im Zuge der Extrusion durch die Verschiebung der ursprünglichen Linie im Raum als Kante des neu erstellten dreidimensionalen Körpers erstellt wird.

Kanten des Typs *project* werden genutzt, um geometrische Elemente von einer Skizze in eine andere Skizze zu projizieren (siehe Abschnitt 5.5.2). Die *project*-Kante zeigt dabei vom Knoten des geometrischen Elements, das in eine andere Skizze projiziert werden soll, auf den Knoten, der das neu zu erstellende geometrische Element repräsentiert. Der Typ des Endknotens dieser Kante muss einer der Typen sein, die entsprechend des folgenden Abschnitts zur Repräsentation von projizierter Geometrie definiert sind.

Knoten- und Kantentypen auf Skizzenebene

In Abbildung 6.22 sind die Knoten- und Kantentypen auf Skizzenebene sowie ihre Attribute definiert. Die entsprechende Definition in der GrGen.NET-*Graph Model Language* ist in Anhang A enthalten.

Mittels der Knoten und Kanten auf Skizzenebene werden parametrische Skizzen repräsentiert. Die verschiedenen Knotentypen dienen zur Repräsentation der geometrischen Elemente (Punkte, Linien, Kreise, Kreisbögen), aus denen sich eine Skizze entsprechend der Beschreibungen in Kapitel 5 zusammensetzen kann. Über die Attribute der geometrischen Elemente werden die zur Erstellung des jeweiligen Elements notwendigen temporären Koordinaten repräsentiert (siehe Abschnitt 5.5). Die Knotentypen, mittels derer die herkömmlichen geometrischen Elemente repräsentiert werden, sind jeweils ein Subtyp des Knotentyps, mit dem das entsprechende projizierte geometrische Element repräsentiert wird. Durch die Nutzung dieses hierarchischen Zusammenhangs wird die Festlegung hinsichtlich der für eine Kante erlaubten Anfangs- und Endknoten vereinfacht. Da z. B. definiert ist, dass eine *mCoincident*-Kante zur Repräsentation der Koinzidenz von Punkten zweier Bauteile immer einen *ProjectedPoint*-Knoten als Anfangs- und Endknoten haben muss, so schließt dies Knoten vom Typ *Point* mit ein. Die Andockstellen werden bei den Knotentypen nicht berücksichtigt, da diese als Attribute der Kanten auf Skizzenebene definiert werden.

Alle nicht abstrakten Kantentypen auf Skizzenebene repräsentieren parametrische Zwangsbedingungen. Dabei repräsentieren Subtypen von *geoConstraint* die geometrischen und Subtypen von *dimConstraint* die dimensionalen Zwangsbedingungen. Durch den abstrakten Kantentyp *portConstraint* wird festgelegt, bei welchen Zwangsbedingungen Andockstellen definiert werden müssen. Die entsprechenden Kantentypen sind in diesem Fall nicht nur

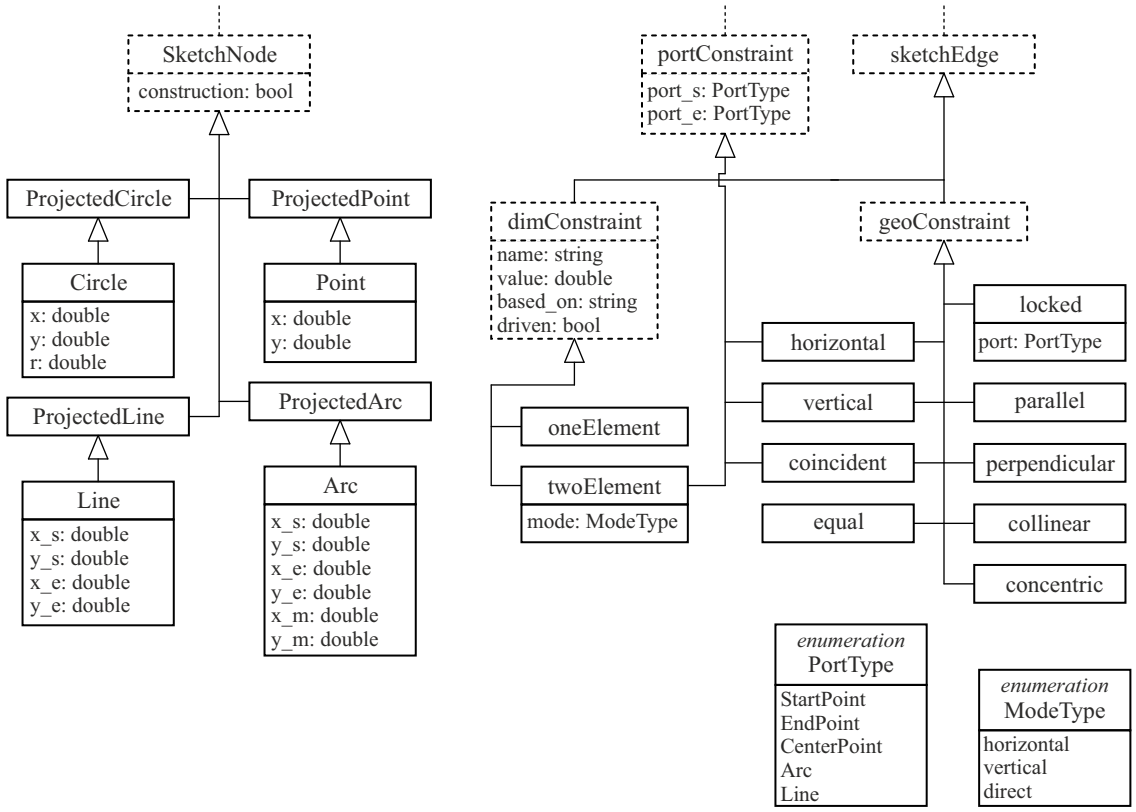


Abbildung 6.22: Knoten- (links) und Kantentypen (rechts) auf Skizzenebene. Hier sind zusätzlich die Attribute, die für diese Typen definiert wurden, inklusive des Datentyps aufgeführt.

Subtypen von *geoConstraint* bzw. *dimConstraint*, sondern auch von *portConstraint* (Mehrfachvererbung). So werden die beiden Attribute *port_s* und *port_e* an diese Typen vererbt. Die Attribute *port_s* und *port_e* können mit Werten aus der Enummeration *PortType* belegt werden, in der alle möglichen Andockstellen definiert sind. Im Attribut *port_s* wird dabei die Andockstelle des geometrischen Elements, das durch den Anfangsknoten der Kante repräsentiert wird, definiert, während *port_e* dies analog für den Endknoten festlegt. Bei der Instantiierung einer Kante durch eine Graphersetzungsregel muss daher durch die Definition der Regel sichergestellt sein, das *port_s* und *port_e* mit Werten belegt werden, die den möglichen Andockstellen der referenzierten geometrischen Elemente entsprechen.

Bei Kanten des Typs *horizontal* oder *vertikal* müssen die *port*-Attribute nur dann belegt werden, wenn mittels der repräsentierten Zwangsbedingung Andockstellen in Form eines Punkts referenziert werden sollen. Dies ist z. B. dann der Fall, wenn der Mittelpunkt eines Kreises horizontal zum Startpunkt einer Linie ausgerichtet werden soll. Werden zwei Punkte (Knotentyp *Point*) horizontal ausgerichtet[26], so müssen die *port*-Attribute nicht belegt werden[27]. Dies ist auch der Fall, wenn eine Linie als horizontal oder vertikal definiert werden soll. In diesem Fall wäre der entsprechende *Line*-Knoten sowohl Anfangs- also auch

[26] Das heißt, dass sie die gleiche y-Koordinate haben.
[27] Sie werden dann mit dem Wert *none* belegt.

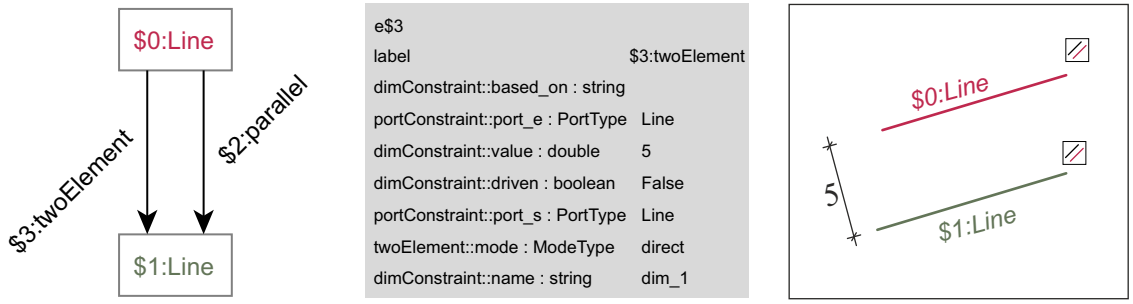


Abbildung 6.23: Graphbasierte Repräsentation (links) zweier paralleler Linien (rechts). Der Abstand der Linien wird durch eine dimensionale Zwangsbedingung auf den Wert 5 festgelegt. Die Attribute der Kante, die diese Zwangsbedingung *$3:twoElement* repräsentiert, sind mittig dargestellt. Hier sind auch die beiden Attribute *port_s* und *port_e* aufgeführt, die definieren, dass sich die Zwangsbedingung auf die beiden Linien als Ganzes (und nicht z. B. auf deren Startpunkt) bezieht.

Endknoten der *horizontal*- bzw. *vertical*-Kante, bei der es sich dann um eine Schleife im Graphen handelt.

Die Repräsentation der Andockstellen im Graphen ist beispielhaft in Abbildung 6.23 dargestellt. Die Abbildung zeigt einen Graphen (links), der zwei parallele Linien (rechts) repräsentiert[28]. Der Abstand zwischen den beiden Linien ist durch eine dimensionale Zwangsbedingung festgelegt. Mittig in der Abbildung sind weiterhin die Attribute der Kante *$3:twoElement* dargestellt. Neben dem Wert und dem Namen eines Attributs ist zusätzlich dargestellt, in welchem Kantentyp dieses Attribut definiert und mittels welchen Datentyps der Wert des Attributs gespeichert ist. Die Attribute *port_s* und *port_e* leiten sich beispielsweise vom (abstrakten) Kantentyp *portConstraint* ab, da der Kantentyp *twoElement* von diesem Kantentyp erbt. Der Datentyp dieser Attribute ist jeweils einer der möglichen Werte der Enummeration *PortType* und in diesem Fall für beide Attribute *Line*. Damit ist definiert, dass sich die dimensionale Zwangsbedingung auf beide Linien als Ganzes bezieht (und nicht z. B. auf deren Startpunkte, siehe dazu auch Abschnitt 5.5.2). Die Parallelität der Linien wird über die Kante *$2:parallel* repräsentiert. Hier ist die Definition von Andockstellen nicht notwendig, da sich diese Zwangsbedingung implizit immer auf die Linien als Ganzes bezieht.

Für die anderen Kantentypen, die von *portConstraint* erben, ist die Repräsentation der Andockstellen analog zum Beispiel im vorigen Absatz umgesetzt. Eine Ausnahme bildet hier der Kantentyp *locked*, der nur ein Attribut zur Definition einer Andockstelle aufweist. Hier ist die Andockstelle notwendig, um zu definieren ob ein geometrisches Element als Ganzes (z. B. eine ganze Linie) oder nur ein Teil des Elements fixiert wird (z. B. ein Start-, Mittel-, oder Endpunkt einer Linie). Generell gelten dabei für die graphbasierte Repräsentation die Festlegungen in Kapitel 5 (dabei insbesondere Abschnitt 5.5.2) hinsichtlich der Verwendung

[28] Es ist dabei zu beachten, dass nur ein Teil der graphbasierten Repräsentation abgebildet ist. Knoten und Kanten auf Bauteil- und Baugruppenebene sind für dieses Beispiel nicht relevant und daher nicht dargestellt.

und Bedeutung der Andockstellen. Im folgenden Abschnitt 6.4.3 ist dies anhand weiterer Beispiele dargestellt.

Neben den Attributen zur Definition der Andockstellen weisen die Kantentypen auf Skizzenebene weitere Attribute auf. Die Kantentypen *oneElement* und *twoElement* erben die Attribute *name*, *value*, *based_on* und *driven* vom abstrakten Kantentyp *dimConstraint*. Mittels des Attributs *name* kann ein Name für eine dimensionale Zwangsbedingung vergeben werden. So kann eine andere dimensionale Zwangsbedingung auf den Wert dieser Zwangsbedingung referenzieren. Eine solche Referenzierung kann in der referenzierenden Zwangsbedingung über das Attribut *based_on* definiert werden. In diesem Fall würde das Attribut *value* der Zwangsbedingung nicht belegt werden. Umgekehrt kann für eine dimensionale Zwangsbedingung ein fester Wert für den durch sie definierten Abstand festgelegt werden, wobei dann das Attribut *based_on* nicht belegt wird.

Das Attribut *driven* ermöglicht es, festzulegen, ob es sich bei einer dimensionalen Zwangsbedingung um eine sogenannte getriebene Zwangsbedingung handelt[29]. In diesem Fall werden die Attribute *value* und *based_on* nicht mit Werten belegt. Über das Attribut *mode*, das nur bei Kanten des Typs *twoElement* vorhanden sein kann, wird festgelegt, wie der durch die Zwangsbedingung definierte Abstand ausgerichtet ist. Über die Enummeration *ModeType* kann entweder der horizontale, der vertikale oder der ausgerichtete (also der kürzeste) Abstand festgelegt werden.

Abschließend ist festzuhalten, dass einige der Kantentypen zur Repräsentation parametrischer Zwangsbedingungen außer dem von *basicEdge* vererbten Attribut *ID* keinerlei Attribute aufweisen. Zur Repräsentation dieser Zwangsbedingungen in einer Skizze ist es ausreichend, dass eine entsprechende Kante erstellt wird, die die Knoten verbindet, mit denen die jeweils betroffenen geometrischen Elemente repräsentiert werden. Werden beispielsweise zwei Knoten des Typs *Circle* durch eine Kante des Typs *concentric* verbunden, so sind die beiden dadurch repräsentierten Kreise als konzentrisch definiert, ohne dass weitere Informationen notwendig sind.

6.4.3 Beispielhafte graphbasierte Repräsentationen

In diesem Abschnitt werden zur Veranschaulichung der konkreten Umsetzung einer graphbasierten Repräsentation verschiedene parametrische Modelle und die entsprechenden Graphen dargestellt. Dazu wird ein Modell schrittweise aufgebaut und damit sowohl der Teil des parametrischen Modells auf Skizzenebene als auch auf Bauteil- und Baugruppenebene beschrieben. Die einzelnen Schritte zum Aufbau des Modells sind in Abbildung 6.24 als

[29] Damit ist eine reine Bemaßung gemeint, die sich aus der durch andere Zwangsbedingungen definierten Topologie einer Skizze ergibt und selbst keinen Einfluss auf den jeweiligen Abstand hat. Eine getriebene Zwangsbedingung kann aber über ihren Namen von einer weiteren dimensionalen Zwangsbedingung referenziert werden

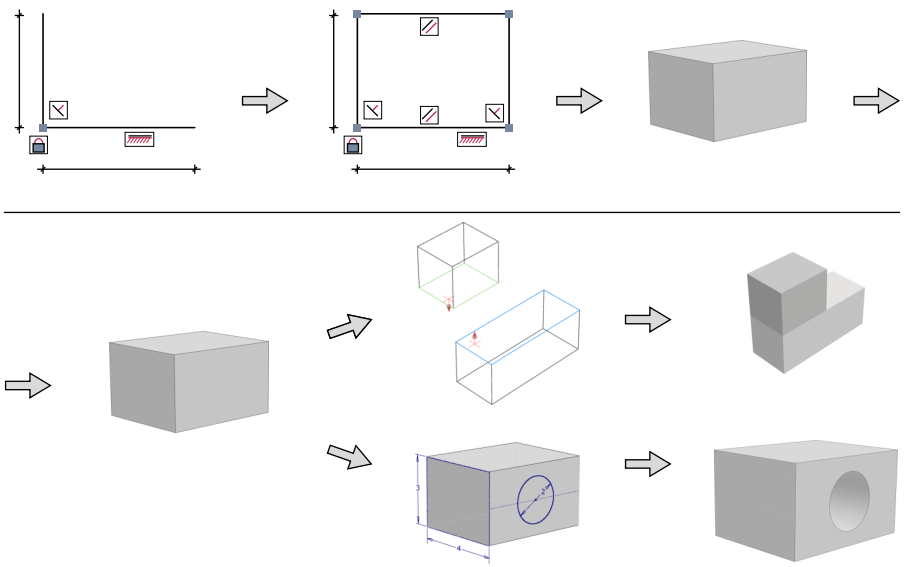


Abbildung 6.24: Überblick über die im Folgenden beschriebenen Beispiele.

Überblick dargestellt. Jeder dieser Schritte ist ein Zustand eines parametrischen Modells, das durch einen Graphen repräsentiert wird und im Folgenden als Beispiel dient. Der grundsätzliche Ablauf beginnt mit der Erstellung der Skizze eines rechteckigen Profils, das anschließend zu einem quaderförmigen Bauteil extrudiert wird. Dieses Bauteil wird nun einerseits mit einem weiteren Quader in einer Baugruppe kombiniert. Zusätzlich wird gezeigt, wie auf einer der Seitenflächen des Quaders eine weitere Skizze angelegt wird, auf deren Basis ein Zylinder erstellt und vom bereits vorhandenen Quader abgezogen wird.

Hierbei liegt der Fokus zuvorderst auf einer möglichst übersichtlichen und verständlichen Erklärung der Umsetzung des Konzepts und nicht auf der Erstellung eines Modells mit praktischem Bezug. Die im Folgenden dargestellten Graphen wurden regelbasiert mittels GrGen.NET erstellt und inklusive ihrer Attribute visualisiert. Um die Beispiele möglichst übersichtlich und nachvollziehbar darzustellen, sind die Abbildungen der einzelnen Zustände des Modells und die entsprechende graphbasierte Repräsentation auf separaten Seiten abgebildet.

In der Regel besteht der erste Arbeitsschritt zur Erstellung eines parametrischen Modells im Anlegen einer Skizze. Abbildung 6.25 zeigt eine solche Skizze, die innerhalb eines Bauteils auf der xy-Arbeitsebene angelegt wurde. Die Skizze besteht aus zwei orthogonalen Linien, deren Länge jeweils durch eine dimensionale Zwangsbedingung festgelegt ist. Weiterhin ist eine Linie innerhalb des Koordinatensystems als horizontal definiert, während der Startpunkt dieser Linie fixiert ist. Das Bauteil, die übergeordnete Baugruppe und die Arbeitsebene sind in der Abbildung des Modells nicht dargestellt, da noch keine 3D-Geometrie vorhanden ist. Neben dem Modell zeigt die Abbildung den Graphen, der das Modell repräsentiert und die Attribute der Knoten und Kanten. Hier ist beispielsweise zu erkennen, dass die

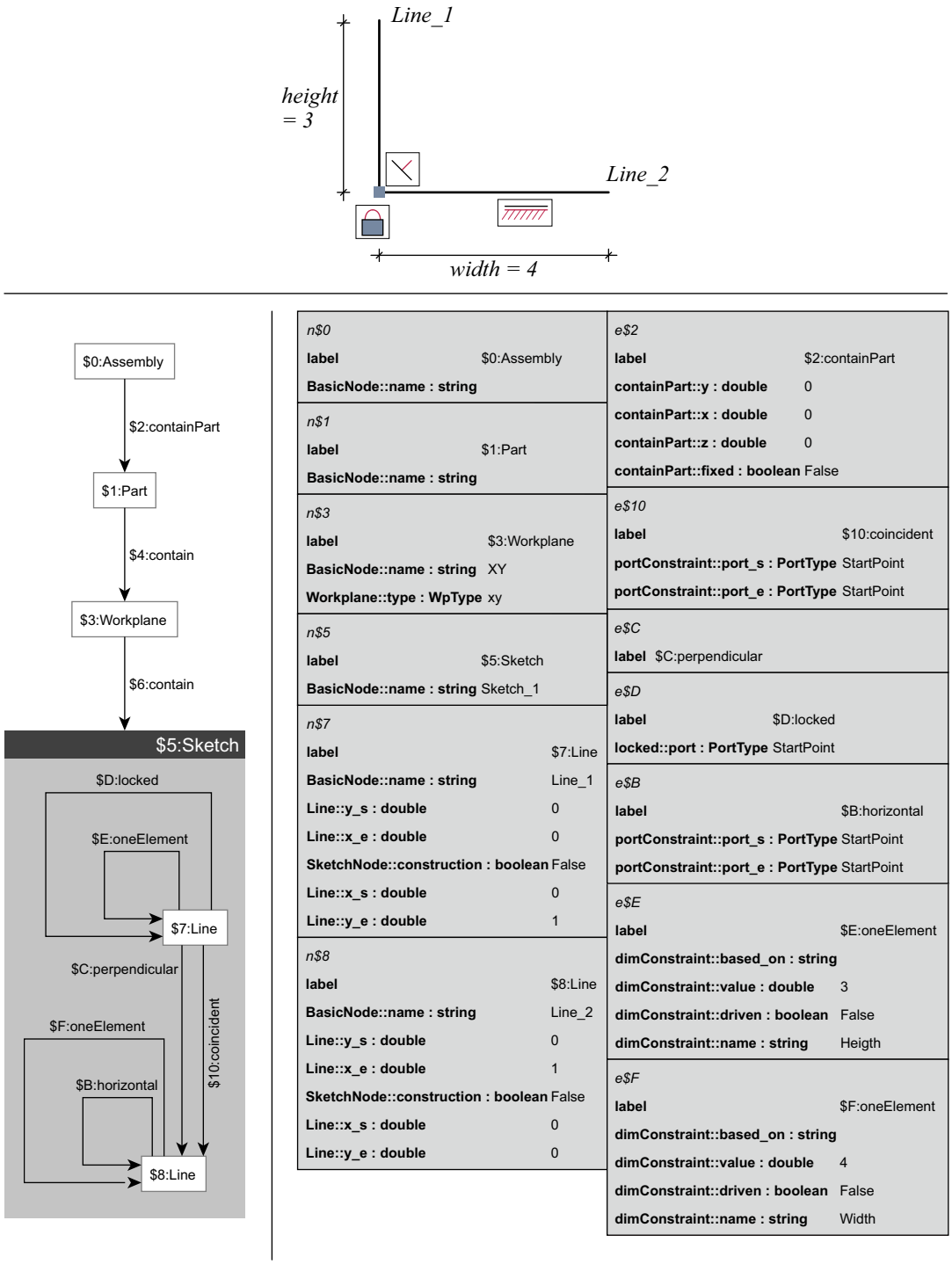
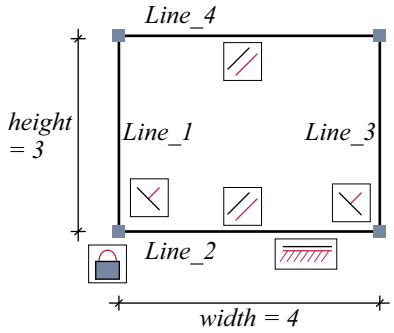
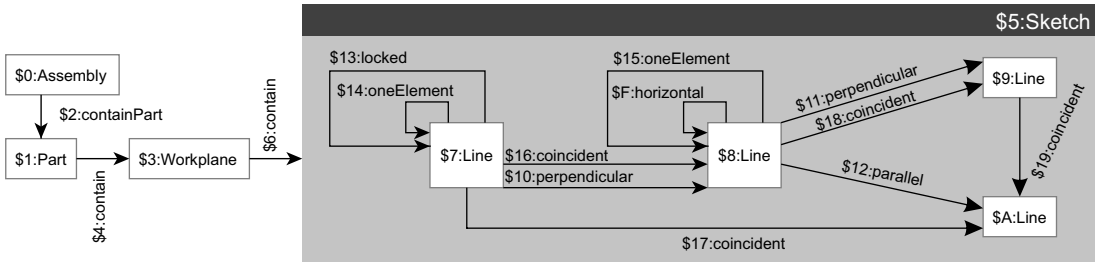


Abbildung 6.25: Graphbasierte Repräsentation (oben links) eines Modells, in dem lediglich eine Skizze, die zwei orthogonale Linien enthält, auf der xy-Arbeitsebene eines Bauteils modelliert ist. Die Attribute der Knoten und Kanten des Graphen sind in den grauen Boxen oben rechts dargestellt. Da die beiden *contain*-Kanten (*e$4*,*e$4*) keine Attribute – außer ihrer *ID*, die hier als *Label* bezeichnet wird – aufweisen, sind sie nicht aufgeführt.





n$7		*n$8*		*n$9*		*n$A*	
label	$7:Line	**label**	$8:Line	**label**	$9:Line	**label**	$A:Line
BasicNode::name : string	Line_1	**BasicNode::name : string**	Line_2	**BasicNode::name : string**	Line_3	**BasicNode::name : string**	Line_4
Line::y_s : double	0	**Line::y_s : double**	0	**Line::y_s : double**	0	**Line::y_s : double**	1
Line::x_e : double	0	**Line::x_e : double**	1	**Line::x_e : double**	1	**Line::x_e : double**	1
...::construction : boolean	False	**...::construction : boolean**	False	**...::construction : boolean**	False	**...::construction : boolean**	False
Line::x_s : double	0	**Line::x_s : double**	0	**Line::x_s : double**	1	**Line::x_s : double**	0
Line::y_e : double	1	**Line::y_e : double**	0	**Line::y_e : double**	1	**Line::y_e : double**	1

e$14		*e$16*		*e$17*	
label	$14:oneElement	**label**	$16:coincident	**label**	$17:coincident
dimConstraint::based_on : string		**...::port_s : PortType**	StartPoint	**...::port_s : PortType**	EndPoint
dimConstraint::value : double	3	**...::port_e : PortType**	StartPoint	**...::port_e : PortType**	StartPoint
dimConstraint::driven : boolean	False				
dimConstraint::name : string	Heigth				

e$18		*e$19*	
label	$18:coincident	**label**	$19:coincident
...::port_s : PortType	EndPoint	**...::port_s : PortType**	EndPoint
...::port_e : PortType	StartPoint	**...::port_e : PortType**	EndPoint

e$15	
label	$15:oneElement
dimConstraint::based_on : string	
dimConstraint::value : double	4
dimConstraint::driven : boolean	False
dimConstraint::name : string	Width

e$F	
label	$F:horizontal
...::port_s : PortType	none
...::port_e : PortType	none

e$13	
label	$13:locked
locked::port : PortType	StartPoint

Abbildung 6.26: Graphbasierte Repräsentation (mittig) eines parametrisch vollständig bestimmten Rechtecks (oben) innerhalb einer Skizze. Es handelt sich hier um eine Erweiterung des Modells aus Abbildung 6.25. Für einige Knoten und Kanten sind die Attribute (unten) nicht aufgeführt, da sie außer der *ID* entweder keine Attribute aufweisen, oder die Attribute in Abbildung 6.25 ersichtlich sind.

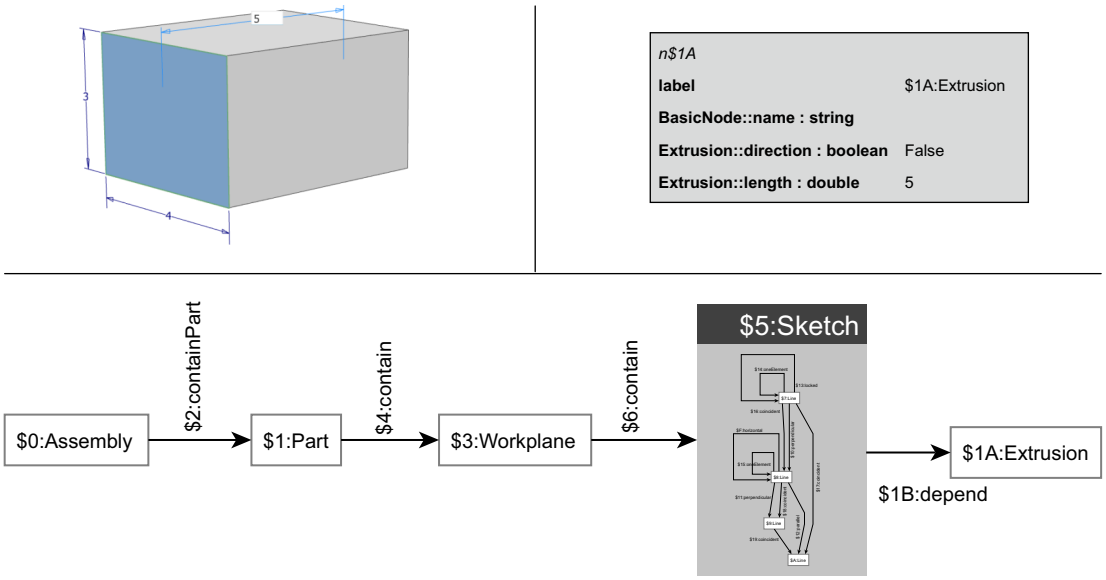


Abbildung 6.27: Extrusion des in der Skizze enthaltenen rechteckigen Profils aus Abbildung 6.26 zu einem Quader. Hierzu ist nur ein weiterer Knoten vom Typ *Extrusion* notwendig, der mittels einer *depend*-Kante mit der Skizze verbunden ist. Über die Attribute dieses Knotens wird die Richtung und die Länge der Extrusion bestimmt.

beiden Linien nicht direkt mit den Längen erstellt werden, die sie durch die dimensionalen Zwangsbedingungen letztlich zugewiesen bekommen. Vielmehr werden die Linien durch die temporären Koordinaten initial so platziert, dass sie korrekt ausgerichtet sind und *Linie_1* nicht z. B. in negative y-Richtung zeigt. Die Startpunkte der beiden Linien sind durch eine entsprechende Zwangsbedingung als koinzident definiert.

Auf Basis der in Abbildung 6.25 eingeführten Skizze ist in Abbildung 6.26 ein erweitertes Modell dargestellt. Hier wurden zwei zusätzliche Linien innerhalb der Skizze ergänzt, die mit fünf weiteren geometrischen Zwangsbedingungen so eingeschränkt werden, dass die vollständig parametrisierte Skizze eines Rechtecks entsteht. Die Abmessungen (Höhe, Breite) dieses Rechtecks sind nun über die beiden schon vorhandenen dimensionalen Zwangsbedingungen steuerbar. An diesem Beispiel lässt sich bereits erkennen, dass bei der Konstruktion einer vergleichsweise einfachen parametrischen Skizze eine Vielzahl von topologischen Abhängigkeiten betrachtet werden muss. Während dies bei einer manuellen Modellierung in den gängigen CAD-Systemen zwar für einzelne Szenarien durch Hilfsfunktionen unterstützt wird, erlaubt es die Nutzung der graphbasierten Repräsentation jedoch, die Erstellung einer beliebigen Skizze[30] als Graphersetzunsgregel zu formalisieren. Da diese Regel dann in verschiedenen Szenarien angewendet werden kann, rechtfertigt sich auch der initiale Aufwand, den die Formulierung der Operation als Graphersetzunsgregel erfordert (siehe folgender Abschnitt 6.5).

[30] Im Rahmen der in Kapitel 5 beschriebenen Funktionen.

Im nächsten Schritt wird auf Basis des in der Skizze erstellen Rechtecks ein Quader durch eine Extrusion erstellt. Dieses Modell und der zugehörige Graph sind in Abbildung 6.27 dargestellt. Hierzu wird lediglich ein *Extrusion*-Knoten mittels einer *depend*-Kante an den *Sketch*-Knoten angehängt. Über das *length*-Attribut des *Extrusion*-Knotens wird die Länge der Extrusion mit dem Wert 5 belegt. Für das Attribut *direction* ist als Wert *false* eingetragen, da die Extrusion in die entgegengesetzte Richtung des Normalenvektors der Skizze (der sich durch die Arbeitsebene bedingt) weist. Auf Basis des so erstellten Körpers werden im Folgenden zwei alternative Szenarien zur weiteren Modellierung aufgezeigt, bei denen die durch die Extrusion neu erstellte Geometrie (Kanten und Flächen des Körpers) referenziert wird.

In Abbildung 6.28 ist das erste dieser beiden Szenarien dargestellt. Das Modell enthält nun ein weiteres quaderförmiges Bauteil, das analog zum bereits beschriebenen Quader modelliert wurde. Lediglich die Länge der Extrusion wurde verdoppelt. Diese beiden Bauteile sind nun über je drei ihrer Seitenflächen aneinander ausgerichtet. Dies ist im Graphen mittels sechs *DerivedGeo*-Knoten und drei *mCoplanar*-Kanten repräsentiert. So werden, wie im oberen Teil der Abbildung dargestellt, verschiedenen Flächen der beiden Bauteile zueinander komplanar ausgerichtet.

In der Abbildung 6.29 ist das zweite Szenario dargestellt. Hier wird auf einer der Seitenflächen des durch Extrusion der Skizze erstellten Quaders eine weitere Skizze erzeugt. Die Arbeitsebene, die dieser Skizze zugrunde liegt, referenziert in der graphbasierten Repräsentation die *Linie_3* in der zuvor erstellten Skizze. Damit ist eindeutig definiert, auf welcher der Seitenflächen des Quaders diese Arbeitsebene erstellt wird. In die neu erstellte Skizze werden die in rot und grün gezeichneten Kanten des Quaders projiziert. Auch diese Kanten beziehen sich auf die *Linie_3*. Die rote Linie ist eine direkte Projektion von *Linie_3*, die grüne Linie eine Projektion der Kante des Quaders, die durch die Verschiebung von *Linie_3* im Raum entstanden ist. Die beiden durch Projektion in die Skizze gezeichneten Linien werden als konstruktive Geometrie definiert.

Um einen Kreis genau im Mittelpunkt der Seitenfläche des Quaders zu platzieren, wird anschließend eine weitere konstruktive Linie $2E erstellt, deren Start- und Endpunkt koinzident zum Mittelpunkt der roten bzw. der grünen Linie sind. Weiterhin ist der Mittelpunkt des Kreises koinzident zum Mittelpunkt dieser Linie $2E. Damit wird die mittige Positionierung des Kreises – unabhängig von den tatsächlichen Koordinaten der Seitenfläche des Quaders – sichergestellt. Durch die erstellten parametrischen Zwangsbedingungen ist die mittige Positionierung auch gegeben, falls die Abmessungen des in der ersten Skizze gezeichneten Rechtecks nachträglich verändert werden.

Die in diesem Abschnitt beschriebenen Beispiele werden im folgenden Abschnitt aufgegriffen, um zu zeigen wie Graphersetzungsregeln zur Konstruktion dieser Modelle formuliert werden können.

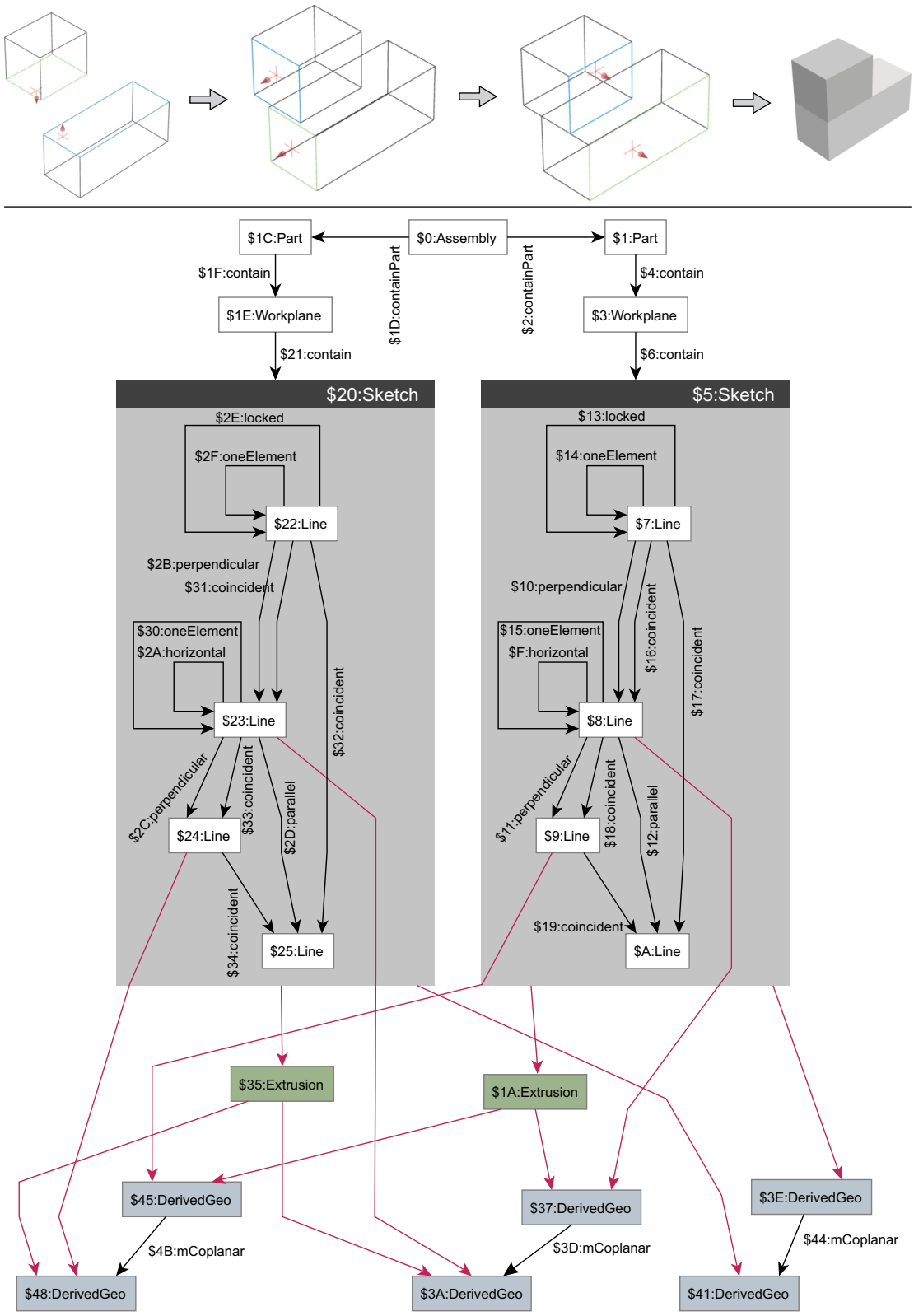
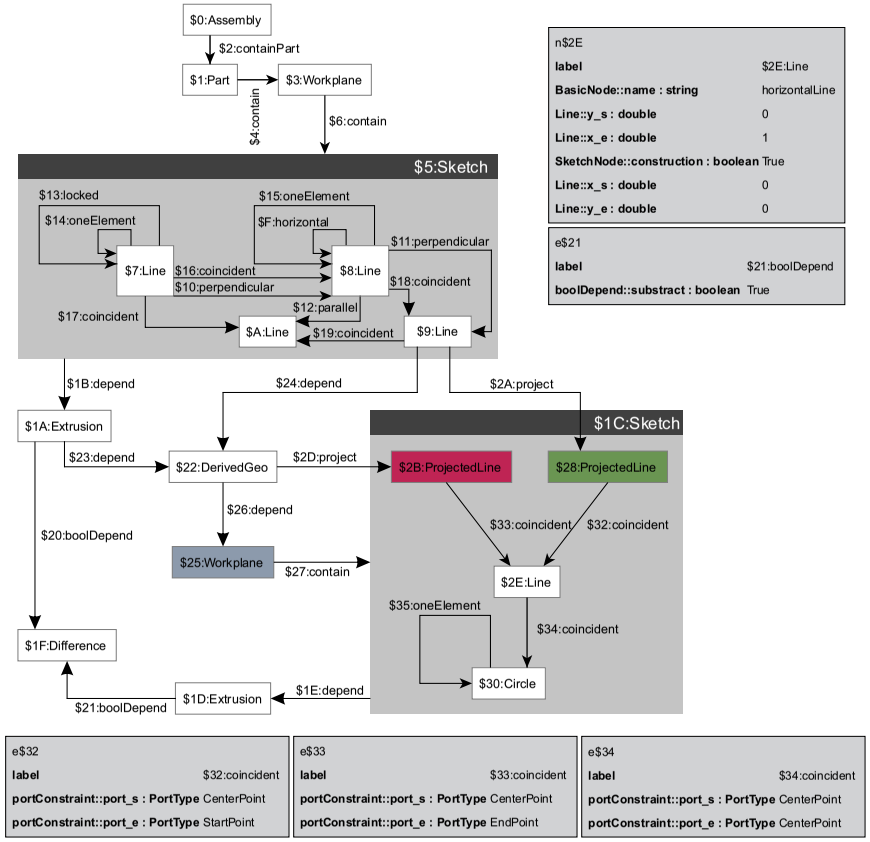


Abbildung 6.28: Modell, das eine Baugruppe enthält, die aus zwei quaderförmigen Bauteilen zusammengesetzt ist. Diese beiden Bauteile sind über je drei ihrer Seitenflächen aneinander ausgerichtet, was im Graphen mittels *DerivedGeo*-Knoten und *mCoplanar*-Kanten repräsentiert wird. Die *depend*-Kanten sind hier in rot gezeichnet und nicht explizit beschriftet.



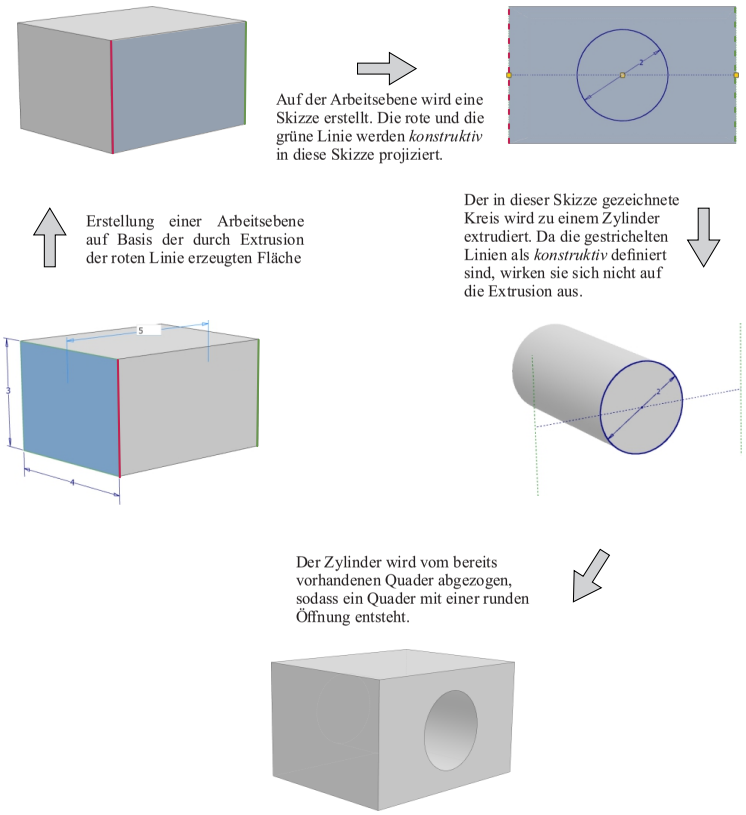


Abbildung 6.29: Rechts sind die Schritte dargestellt, mit denen auf einer der Seitenflächen des Quaders die Skizze eines Kreises erstellt wird. Der durch anschließende Extrusion erstellte Zylinder kann danach wiederum vom ursprünglichen Quader abgezogen werden. Links ist die graphbasierte Repräsentation des resultierenden Modells dargestellt.

6.5 Graphersetzungsregeln

Graphersetzungsregeln werden im hier vorgestellten Ansatz verwendet, um Modellierungs*schritte* sowie Modellierungs*abläufe* zu formalisieren. Durch die konsekutive Ausführung mehrerer Graphersetzungsregeln kann so ein Modellierungs*prozess* beschrieben werden. Diese Begriffe und ihre Zusammenhänge sind für die Verwendung im weiteren Verlauf dieser Arbeit wie folgt definiert[31] und in Abbildungen 6.30 und 6.31 dargestellt:

- Der Begriff Modellierungsschritt beschreibt genau einen singulären Vorgang, mit dem der Zustand eines Modells bearbeitet und dadurch verändert oder erweitert wird. Dies ist das Ausführen einer Funktion eines CAD-Werkzeugs wie z. B. das Erstellen oder Löschen eines geometrischen Objekts oder einer parametrischen Zwangsbedingung bzw. das Ausführen einer prozeduralen Operation.

- Ein Modellierungsablauf ist eine Reihe von konsekutiven Modellierungsschritten, die in ihrer Gesamtheit eine höherwertige Veränderung eines Modells verursachen. Ein Modellierungsablauf ist beispielsweise das Erstellen eines Rechtecks durch die Modellierung von vier Linien, die mittels parametrischer Zwangsbedingungen angeordnet werden.

- Ein Modellierungsprozess umfasst eine Abfolge von verschiedenen Modellierungsschritten und/oder Abläufen, die eine wesentliche konzeptionelle Weiterentwicklung während eines modellbasierten Entwurfsprozesses beschreibt. Dies kann z. B. die Detaillierung eines kompletten Bauteils sein.

- Weiterhin wird der Begriff Modellierungs*operation* verwendet, um die Veränderung eines Modells generell zu beschreiben, wenn es nicht notwendig ist näher zu spezifizieren, ob ein Modellierungsschritt, ein Modellierungsablauf oder ein Modellierungsprozess gemeint ist.

In den meisten in dieser Arbeit beschriebenen Beispielen wird mittels einer Graphersetzungsregel ein Modellierungsablauf repräsentiert. In diesem Fall beschreibt die Graphersetzungsregel eine Reihe von Modellierungsschritten, die in ihrer Gesamtheit die Bearbeitung eines Modells zu einem bestimmten Zweck beschreiben, wie beispielsweise das mittige Platzieren eines Kreises auf der Seitenfläche eines Quaders (siehe Beispiel im vorigen Abschnitt).

Grundsätzlich kann eine Graphersetzungsregel aber auch zur Repräsentation lediglich eines einzelnen Modellierungsschritts verwendet werden. Dies ist allerdings nur in bestimmten Fällen sinnvoll, da das manuelle Ausführen eines einzelnen Modellierungsschritts im Normalfall

[31] Es handelt sich hierbei nicht um allgemeingültige Definitionen. Im Kontext der Inhalte insbesondere dieses Abschnitts ist eine genaue Definition jedoch von Vorteil.

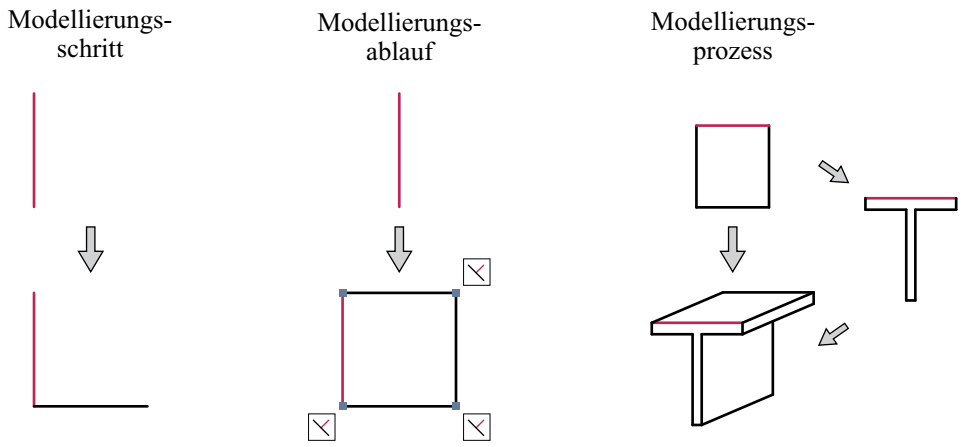


Abbildung 6.30: Beispiele für einen Modellierungsschritt, einen Modellierungsablauf und einen Modellierungsprozess. Für den Modellierungsprozess ist zusätzlich ganz rechts dargestellt, dass dieser sich aus zwei Modellierungsabläufen zusammensetzt.

weniger aufwändig ist, als eine entsprechende Graphersetzungsregel zu formulieren. Wird ein Modellierungsprozess jedoch in Abläufe und Schritte zerlegt, die durch eigenständige Regeln beschrieben und dadurch in anderen Prozessen wiederverwendet werden können, kann es sinnvoll sein, auch einzelne Modellierungsschritte als Graphersetzungsregeln zu formalisieren. Ein Modellierungsprozess ist daher immer ein Tupel aus Graphersetzungsregeln, die sequenziell ausgeführt werden. Abbildung 6.31 stellt die Zusammenhänge zwischen Modellierungsschritten, -abläufen und -prozessen im Kontext der Repräsentation durch Graphersetzungsregeln dar.

Der wesentliche Unterschied zwischen Graphersetzungsregeln, die einen Modellierungsschritt bzw. einen Modellierungsablauf repräsentieren ist, dass in einer Graphersetzungsregel, die einen Modellierungsablauf repräsentiert, zwar das Resultat der Ausführung mehrerer Modellierungsschritte transportiert wird, aber nicht zwangsläufig die Schritte, die tatsächlich manuell zur Konstruktion ausgeführt werden müssten (z. B. das Anlegen temporärer Konstruktionslinien). Dies ist beispielsweise bei dem ersten Modellierungsablauf des in

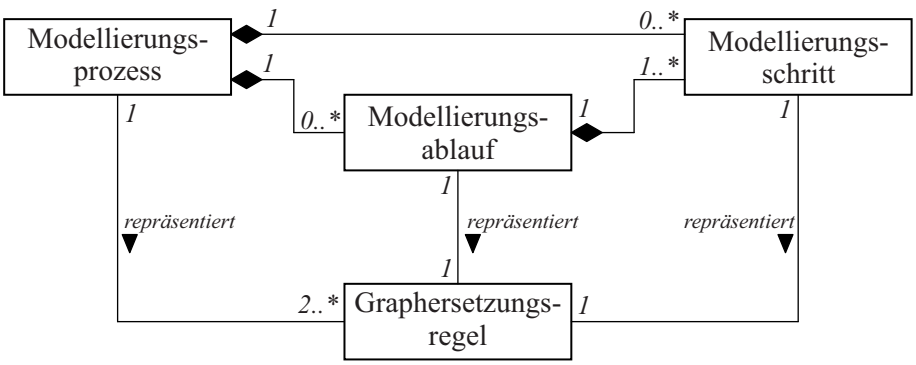


Abbildung 6.31: Zusammenhang zwischen Modellierungsoperationen und Graphersetzungsregeln

Abbildung 6.30 rechts dargestellten Modellierungsprozesses der Fall. Hier wird direkt aus einem Rechteck der Querschnitt eines T-Trägers erstellt, ohne dass die einzelnen Modellierungsschritte, die bei einer manuellen Modellierung nötig wären, explizit beschrieben werden.

Im folgenden Abschnitt wird das Vorgehen zur Formalisierung von Modellierungsschritten, Modellierungsabläufen und Modellierungsprozessen erklärt und anhand von Beispielen beschrieben.

6.5.1 Definition und Aufbau von Graphersetzungsregeln zur Repräsentation von Modellierungsoperationen

Der rein formale Aufbau von Graphersetzungsregeln wurde bereits in Kapitel 4 ausführlich beschrieben. Um Graphersetzungsregeln zu konzipieren, die ein Modell verändern, indem der Graph verändert wird, der dieses Modell entsprechend den in diesem Kapitel getroffenen Festlegungen repräsentiert, muss die graphbasierte Repräsentation dieses Modells vor und nach der beabsichtigten Veränderung betrachtet werden. Hierzu muss zuerst rein auf Modellebene untersucht werden, welche Veränderung beabsichtigt ist, um anschließend eine graphbasierte Repräsentation und damit eine Graphersetzungsregel ableiten zu können. Diese Schritte sind im Folgenden beschrieben:

- Definition einer bestimmten Modellierungsoperation. Dazu muss der Bestandteil des Modells, der verändert wird, hinsichtlich seines Ist- und seines Sollzustandes betrachtet werden, um zu bestimmen, welche einzelnen Schritte notwendig sind.
- Ableitung der graphbasierten Repräsentation des Ist- und des Sollzustandes des betrachteten Teils eines Modells. Dabei ist zu beachten, dass ein ausreichender Kontext[32] vorhanden ist, um z. B. neu erstellte Objekte an der korrekten Stelle in ein Gesamtmodell einfügen zu können oder genau einen bestimmten Teil eines Modells zu verändern.
- Durch die Gegenüberstellung der graphbasierten Repräsentationen des Ist- und Sollzustands kann nun eine Graphersetzungsregel konzipiert werden, die Knoten und Kanten hinzufügt, entfernt oder deren Attribute verändert. Die Graphen, die diese Zustände repräsentieren, entsprechen dem Arbeitsgraphen und dem modifizierten Arbeitsgraphen einer Graphtransformation.

[32] Der Kontext einer Modellierungsoperation ist der relevante Teil des Ausgangszustands eines Modells, das verändert werden soll. Über den Kontext wird sichergestellt, dass die Operation nur in bestimmten Teilen eines Modells angewendet werden kann. In einer Graphersetzungsregel wird dieser Kontext durch den Mustergraphen repräsentiert.

In den Beispielen im folgenden Abschnitt wird die Konzeption verschiedener Graphersetzungsregeln anhand dieser Schritte beschrieben. Dabei werden die Modellierungsoperation, die repräsentierenden Graphen und die Graphersetzungsregeln zuerst visuell dargestellt. Zusätzlich wird die formale Definition der Graphersetzungsregel in der GrGen.NET *Rule and Computations Language* beschrieben, da insbesondere die Veränderung von Attributen visuell kaum übersichtlich dargestellt werden kann.

Im ersten Teil dieses Kapitels wurde bereits die Definition des Graph-Metamodells eingeführt. Diese schließt auch mit ein, inwiefern das Metamodell in einer direkten Beziehung zur Entwicklung von Graphersetzungsregeln steht, und wie sich diese beiden Teile des Graphersetzungssystems gegenseitig beeinflussen. In diesem Abschnitt werden nun die generellen Anforderungen, die sich an die Graphersetzungsregeln stellen, erläutert.

Die wesentliche Anforderung an eine Graphersetzungsregel, mit deren Hilfe eine Modellierungsoperation repräsentiert wird, besteht darin, dass die Anwendung dieser Graphersetzungsregel genau die durch den Nutzer beabsichtige Veränderung des Modells zur Folge hat. Eine Graphersetzungsregel muss also hinsichtlich ihrer Beschreibung und ihrer formalen Definition eindeutig formuliert sein. Die Beschreibung schließt dabei mit ein, in welchen Modellierungsszenarien die Graphersetzungsregel angewendet werden kann, also wann ein Nutzer eine bestimmte Regel verwenden kann. Hierbei ist zu beachten, dass bei der Ausführung einer Graphersetzungsregel nur sichergestellt ist, dass ein bestimmter Mustergraph und damit ein bestimmter Modellzustand vor der Ausführung gegeben ist und dieser Mustergraph auch formal korrekt verändert wird. Ob durch die Anwendung der Regel aber eine im jeweiligen Szenario sinnvolle und fachlich richtige Modellierungsoperation durchgeführt wird, muss der Nutzer entscheiden, der die Ausführung der Regel aktiv veranlasst. Es können zwar durch den im Mustergraphen definierten Kontext Randbedingungen definiert werden – allerdings können diese beispielsweise nicht verhindern, dass eine fachlich falsche Veränderung des Modells durch den Nutzer veranlasst wird. Damit liegt trotz der durch die Regel abgebildeten Automatisierung der Modellierung die Entscheidung letztlich beim ausführenden Ingenieur. Prinzipiell ist es aber auch möglich eine Menge an Graphersetzungsregeln zu entwickeln, die ausschließlich geometrisch und fachlich korrekte Modelle erzeugen – allerdings führt dies zu einem eingeschränkten Raum an möglichen Entwürfen, der dementsprechend nur einzelne Entwurfsszenarien abdecken kann.

Neben der korrekten Repräsentation des Kontexts einer Modellierungsoperation und die durch eine Modellierungsoperation beabsichtige Veränderung, muss eine Graphersetzungsregel auch den durch das Graph-Metamodell vorgegebenen Definitionen entsprechen. Dies sind einerseits die formalen Definitionen, mit denen festgelegt wird, wie der Graph aufgebaut sein kann. Eine Graphersetzungsregel darf also kein Ergebnis produzieren, das Knoten- oder Kantentypen enthält, die nicht Teil des Metamodells sind. Das gleiche gilt für Attribute und die Struktur des Graphen. Knoten dürfen also nur durch bestimmte

Kanten verbunden sein und die Werte von Attributen müssen den definierten Datentypen entsprechen. Die Einhaltung dieser Anforderungen kann automatisch durch GrGen.NET sichergestellt werden. Wird eine in der *Rule and Computations Language* definierte Regel kompiliert, führen Widersprüche zum Graph-Metamodell direkt zu einem Fehler. Weiterhin kann der durch eine Ersetzungsregel erzeugte Graph auf die Einhaltung der Vorgaben zur Struktur (erlaube Verbindungen von Knoten und Kanten) überprüft werden. Neben den rein formalen Vorgaben müssen auch die in Abschnitt 6.2 beschriebenen konzeptionellen Vorgaben und Konventionen eingehalten werden, damit eine Interpretation des Graphen fehlerfrei möglich ist. Beispielhaft seien hier die korrekte Verwendung von Andockstellen und temporären Koordinaten genannt. Werden diese Konventionen bei der Erstellung von Ersetzungsregeln nicht eingehalten, so führt eine automatisierte Interpretation des Graphen auf Basis dieser Regeln zwangsläufig zu einem fehlerhaften Ergebnis. Für die Erstellung von Ersetzungsregeln ist es daher zwingend notwendig, diese Vorgaben und Konventionen vollständig zu kennen und anzuwenden. Der Regelerstellungsprozess bleibt damit Nutzern mit entsprechender Expertise vorbehalten und ist je nach Komplexität der Regel mit einem entsprechenden Aufwand verbunden. Zu beachten ist dabei allerdings, dass dies für die reine Anwendung der Regeln nicht der Fall ist.

Generell steigt die Komplexität einer Graphersetzungsregel und damit einhergehend der Aufwand bei der Erstellung mit dem Umfang des Modellierungszenarios, in dem sie angewendet werden kann. Regeln, die in einem konkreten Modellierungsszenario angewendet werden sollen, sind vergleichsweise einfach zu definieren. Allerdings sind die Anwendungsmöglichkeiten dann entsprechend eingeschränkt und die Anwendung der Regel in einem leicht veränderten Szenario wäre nur durch eine Anpassung der Regel realisierbar. Andererseits ist es auch möglich Graphersetzungsregeln zu definieren, die in einem vielfältigen Kontext angewendet werden können. Die GrGen.NET *Rule and Computations Language* ermöglicht es beispielsweise, auch bedingte Anweisung und Verzweigungen in einer Graphersetzungsregel abzubilden. Damit kann eine Regel so angepasst werden, dass sie je nach der genauen Beschaffenheit des Mustergraphen den Ersetzungsgraphen anpasst. Neben dem erhöhten Aufwand bei der Erstellung einer solchen Regel, muss dieses Verhalten auch an den Nutzer kommuniziert werden. Letztlich lässt sich festhalten, dass der Aufwand bei der Erstellung einer Regel mit der Komplexität des Modellierunsszenarios und einer vielseitigeren Anwendungsmöglichkeit steigt, was aber mit einem entsprechend gesteigerten Nutzen einhergehen kann.

6.5.2 Beispiele

Im Folgenden wird das Vorgehen zur Erstellung von Graphersetzungsregeln zur Repräsentation von Modellierungsschritten, -abläufen und -prozessen anhand mehrerer Beispiele detailliert beschrieben. Dabei wird sowohl die jeweilige Modellierungsoperation graphisch

in einem Modell dargestellt als auch der für die Repräsentation relevante Teil des entsprechenden Graphen. Daraus wird letztlich die eigentliche Graphersetzungsregel abgeleitet und visuell durch den Muster- und den Ersetzungsgraphen beschrieben. Zusätzlich wird die formale Beschreibung der Regel in der GrGen.NET *Rule and Computations Language* definiert.

Die hier dargestellten einfachen und übersichtlichen Beispiele bilden die Grundlage für das Verständnis der Fallstudien im folgenden Kapitel, bei denen eine Visualisierung der Ersetzungsregeln und der graphbasierten Repräsentationen teils nicht mehr übersichtlich möglich und daher im Anhang aufgeführt ist.

Graphersetzungsregeln zur Repräsentation von Modellierungsschritten

In ersten Minimalbeispiel wird ein Modellierungsschritt zur Erstellung einer Linie innerhalb einer bereits bestehenden Skizze gezeigt. Es handelt sich hierbei um einen vergleichsweise trivialen Modellierungsschritt – letztlich setzt sich aber jeder Modellierungsablauf und jeder Modellierungsprozess aus genau solchen Einzelschritten zusammen, die erst durch ihre Kombination zu einem komplexen Vorgang werden. Dies gilt im gleichen Maß für die Erstellung entsprechend komplexer Graphersetzungsregeln, die genauso aus weniger komplexen Einzelteilen zusammengesetzt werden können. Zur Erstellung bietet es sich daher an, eine Bibliothek von Regeln vorzuhalten, die als Vorlage dienen und je nach Bedarf wiederverwendet und dabei gegebenenfalls abgeändert werden können. So kann der Aufwand für den Nutzer bei der Erstellung einer neuen Regel reduziert werden, sofern bereits Regeln vorhanden sind, die für ähnliche Modellierungsszenarien konzipiert wurden.

In Abbildung 6.32 ist zuoberst dargestellt, wie sich ein geometrisches Modell durch die Anwendung einer Graphersetzungsregel verändert. Auf der linken Seite sind hier die drei Koordinatenebenen eines Bauteils *$0:Part* dargestellt, wobei auf der yz-Ebene *$1:Workplane* bereits eine Skizze *$3:Sketch* erstellt wurde, was durch die im mittleren Bereich der Abbildung dargestellte graphbasierte Repräsentation verdeutlicht wird[33]. Das rechts dargestellte Resultat enthält nun, sowohl im Modell als auch im Graphen, eine neu erstellte Linie *$5:Line*, die auf dieser Skizze liegt. Dass diese Linie der Skizze zugeordnet wird, ist visuell durch die Darstellung als Gruppe implizit definiert (vergl. Abschnitt 4.1.2). In Listing 6.3, Zeile 12 wird dies in der Graphersetzungsregel explizit definiert, da die Skizze und die Linie mittels einer *contain*-Kante verbunden werden. Den Beschriftungen, durch die die Typen der Knoten und Kanten gekennzeichnet sind, ist ein durch *$* gekennzeichneter Identifikator vorangestellt[34].

[33] Dass es sich bei dieser Arbeitsebene um die yz-Ebene handelt, wird über die Attribute von *$1:Workplane* definiert, die hier nicht dargestellt sind.

[34] Dieser Identifikator wird in der Definition des Graph-Metamodells als *ID* bezeichnet. Diese IDs werden bei der Erstellung von Graphen in GrGen.NET durch hexadezimale Nummerierung vergeben.

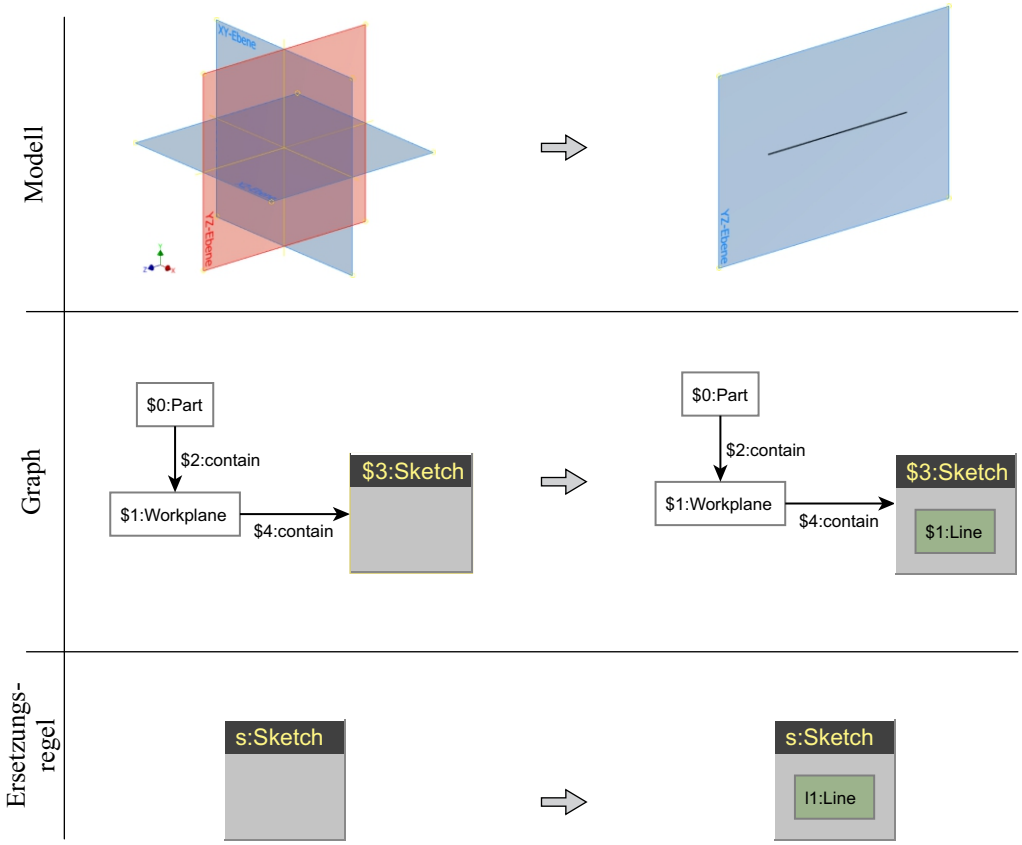


Abbildung 6.32: Im hier dargestellten Modellierungsschritt wird eine Linie innerhalb einer bereits bestehenden Skizze durch die Regel *CreateLine* erstellt. Bei der Darstellung der Graphen ist die Passung des Mustergraphen im Arbeitsgraphen jeweils gelb markiert. Rechts ist der durch die Anwendung der Regel neu hinzugefügte Knoten in grün dargestellt.

Im unteren Bereich der Abbildung 6.32 ist nun die eigentliche Graphersetzungsregel visuell dargestellt. Hierbei ist anzumerken, dass diese Regel als Kontext lediglich eine Skizze beinhaltet. Mit dieser Regel kann also in jeder beliebigen Skizze eines Modells eine Linie erzeugt werden. Die Regel erstellt entsprechend einen neuen *Line*-Knoten, der durch eine *contain*-Kante mit dem *Sketch*-Knoten im Mustergraphen verbunden wird. Dass es sich sowohl im Muster- als auch im Ersetzungsgraphen um denselben *Sketch*-Knoten handeln muss, wird durch das vorangestellte *s:* gekennzeichnet[35]. Offensichtlich handelt es sich bei dieser Regel also um eine sehr generische Definition eines Modellierungsschritts, die einerseits vergleichsweise trivial ist, aber andererseits in einer Vielzahl von Modellierungsszenarien als leicht anpassbare Vorlage verwendet werden kann.

[35]Konkret wird hierdurch ein Erhaltungsmorphismus definiert.

```
1  rule CreateLine{
2      s:Sketch;
3  modify{
4      l1:Line;
5      eval{
6          l1.x_s = -2;
7          l1.y_s = 1;
8          l1.x_e = 2;
9          l1.y_e = 1;
10         l1.name = "Line1";
11         }
12     s-:contain->l1;
13 }}
```

Listing 6.3: Formale Definition der in Abbildung 6.32 visuell dargestellten Graphersetzungsregel mittels der *Rule and Computations Language* von GrGen.NET.

Zusätzlich zur Visualisierung als Graph ist die Regel in Listing 6.3 mittels der *Rule and Computations Language* von GrGen.NET beschrieben. Hierbei handelt es sich um die exakte Definition der Regel, die auch die Attribute von Knoten und Kanten mit einschließt, da diese in der visuellen Darstellung nicht optimal abgebildet werden können. In Zeile 1 wird mit *CreateLine* der Name der Regel definiert. In Zeile 2 wird hier der Mustergraph definiert, der in diesem Fall nur einen Knoten *s:Skizze* beinhaltet, der im Folgenden mit dem Identifikator *s* referenziert werden kann. Das Statement *modify* in Zeile 3 leitet die Definition des Ersetzungsgraphen ein. Mit *l1:Line* wird dem Mustergpraphen ein neuer *Line*-Knoten hinzugefügt, der als Identifikator *l1* erhält, um ihn referenzieren zu können. Die Identifikatoren kommen in Zeile 12 zum Einsatz, in der definiert wird, dass der *Sketch*-Knoten *s* aus dem Mustergraphen und der im Ersetzungsgraphen neu definierte *Line*-Knoten *l1* mittels einer *contain*-Kante verbunden werden. Durch die Referenzierung des *Sketch*-Knotens *s* im Ersetzungsgraphen wird weiterhin festgelegt, dass dieser Knoten im Zuge der Ersetzung erhalten bleibt und nicht gelöscht wird.

Um die initiale Postionierung der Linie festzulegen, müssen zusätzlich die temporären Koordinaten der Linie mit Werten belegt werden. Dies wird über das *eval*-Statement in Zeile 5 eingeleitet und in den folgenden Zeilen 6 bis 9 definiert. Neben den temporären Koordinaten des Start- und des Endpunkts wird für die Linie weiterhin der Name *Linie1* in Zeile 10 festgelegt. Durch diesen Namen kann in weiteren Graphersetzungsregeln im Mustergraphen definiert werden, dass eine Passung nur mit *Line*-Knoten, die genau diesen Namen haben, gefunden wird. Dies ermöglicht es, Regeln zu definieren, die nur auf spezifische geometrische Elemente angewendet werden können, und es damit ermöglichen, Änderungen an genau diesen geometrischen Elementen vorzunehmen. Weiterhin können verschiedene Bauteile mittels solcher benannter geometrischer Elementen in einer Baugruppe aneinander ausgerichtet werden.

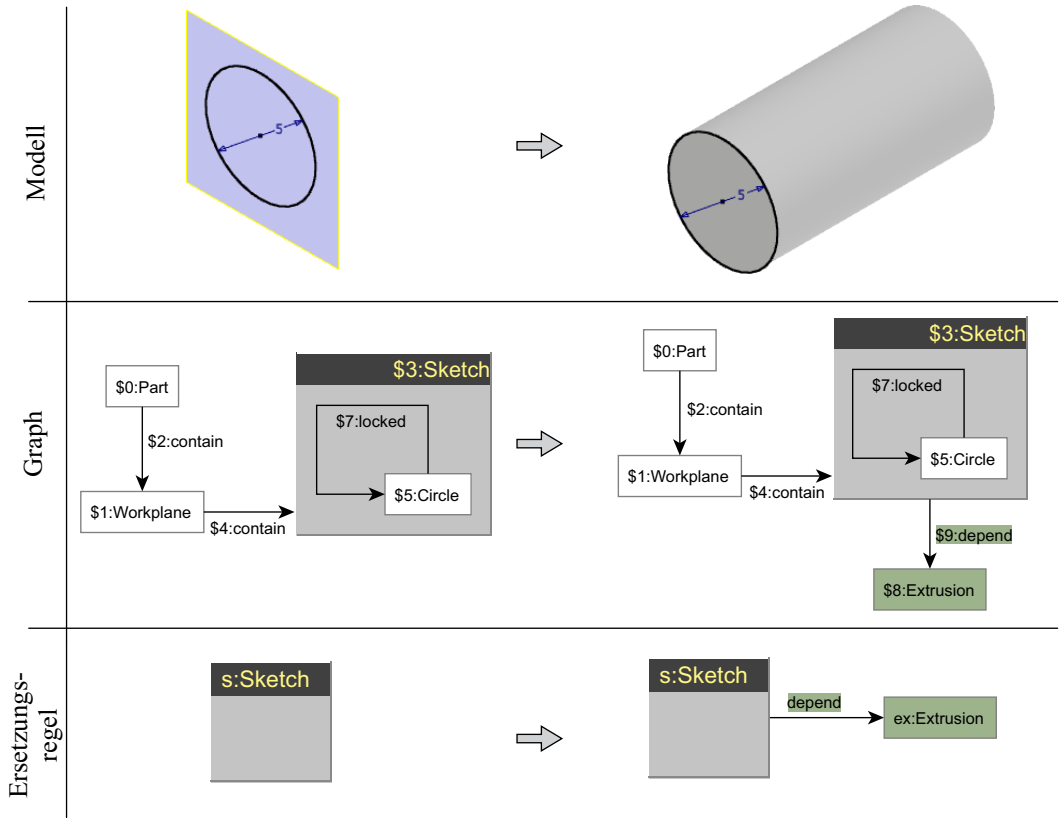


Abbildung 6.33: Im dargestellten Modellierungsschritt wird auf Basis einer Skizze eine Extrusion erzeugt, was mittels der Graphersetzungsregel *Extrude* repräsentiert wird.

Als zweites Beispiel eines Modellierungsschritts wird die in Abbildung 6.33 dargestellte Extrusion herangezogen. Als Basis dient eine Skizze, die einen Kreis beinhaltet, sodass durch die Extrusion ein Zylinder entsteht. Auch in diesem Beispiel zeigt die Abbildung zuerst die beabsichtige Veränderung des Modells und danach die graphbasierte Repräsentation des Modells vor und nach der Anwendung der Regel.

Die Graphersetzungsregel ist neben der visuellen Darstellung in Listing 6.4 formal definiert. Hier ist anzumerken, dass es sich um eine äußert generische Regel handelt, mit der jegliche Skizze unabhängig von ihrem Inhalt als Basis einer Extrusion dienen kann. Damit kann diese Regel potenziell in einer Vielzahl von Modellierungsprozessen verwendet werden.

```
rule Extrude(var l:double){
    s:Sketch;
modify{
    s-:depend->ex:Extrusion;
    eval{
        ex.length = l;
    }
}}
```

Listing 6.4: Formale Definition der in Abbildung 6.33 visuell dargestellten Graphersetzungsregel mittels der *Rule and Computations Language* von GrGen.NET.

Um die Länge der Extrusion bei der Verwendung dieser Regel nicht im Code anpassen zu müssen, ist für diese Regel ein Parameter l definiert, über den die Länge der Extrusion gesteuert werden kann. Bei der Ausführung der Regel muss für diesen Parameter ein Wert angegeben werden.

```
rule ExtrudeFromSketch3D{
    s:Sketch;
    s3d:Sketch3D;
    p1:Point3D; p2:Point3D; p1-:depend->p2;
    s3d-:contain->p1; s3d-:contain->p2;
    s3d-:depend->wp:Workplane-:contain->s;
    p1-:depend->wp;
    negative {:Point3D-:depend->p1-:depend->p2;}
    negative {p1-:depend->p2-:depend->:Point3D;}
modify{
    s-:depend->ex:Extrusion;
    eval{
        ex.length = Math::sqrt(Math::pow((p2.x-p1.x),2.0)+Math::pow((p2.y-
            p1.y),2.0)+(Math::pow((p2.y-p1.y),2.0)));
    }
}}
```

Listing 6.5: Formale Definition einer Graphersetzungsregel durch die eine 2D-Skizze entsprechend der Länge einer Linie, die in einer 3D-Skizze enthalten ist, extrudiert wird.

Eine Alternative, bei der die Länge der Extrusion dynamisch während der Graphersetzungsoperation berechnet wird, ist in Listing 6.5 dargestellt. Ausgangspunkt sind hier eine 2D-Skizze und eine 3D-Skizze innerhalb eines Bauteils. Die 2D-Skizze muss dabei auf einer Arbeitsebene erstellt worden sein, die wiederum auf Basis der 3D-Skizze erstellt wurde. Gemäß der Definitionen in diesem Kapitel bedeutet dies, dass die Arbeitsebene orthogonal zum Verlauf der 3D-Skizze auf einem der Punkte innerhalb der 3D-Skizze erstellt wurde. Die Regel extrudiert nun die 2D-Skizze entsprechend der Länge der durch die 3D-Skizze definierten Linie, indem der Abstand von Anfangs- und Endpunkt innerhalb der Regel

berechnet wird (Zeile 13). Dieser Modellierungsschritt kann beispielsweise Teil eines Modellierungsprozesses sein, mit dem die Darstellung eines Trägers von einem abstrahierten Stab zur Extrusion des Profils detailliert wird. Der Inhalt der Skizze müsste in diesem Szenario natürlich über eine separate Regel erstellt werden. Zu beachten ist, dass bei dieser Regel in den Zeilen 7 und 8 negative Anwendungsbedingungen definiert werden. So wird sichergestellt, dass diese Regel nur dann angewendet werden kann, wenn die 3D-Skizze durch genau zwei Punkte als Linie definiert wird. Enthält eine 3D-Skizze mehr als zwei Punkte, greift die negative Anwendungsbedingung und es wird keine Passung des Mustergraphen im Arbeitsgraphen gefunden.

Graphersetzungsregeln zur Repräsentation von Modellierungsabläufen

Im vorangegangenen Abschnitt wurden die graphbasierten Repräsentationen verschiedener Modellierungsschritte vorgestellt. Dabei wurde auch das grundsätzliche Vorgehen zur Beschreibung einer Regel sowohl visuell als Graph und textuell in der *Rule and Computations Language* von GrGen.NET eingeführt. Darauf aufbauend werden im Folgenden drei Graphersetzungsregeln zur Repräsentation von Modellierungsabläufen dargestellt und beschrieben.

In Abbildung 6.34 wird zuerst ein Modellierungsablauf dargestellt, der eine bestehende Linie als horizontal definiert und die Länge dieser Linie mittels einer dimensionalen Zwangsbedingung auf den Wert 4,0 festlegt. Hierbei handelt es sich um zwei Modellierungsschritte, die in diesem Modellierungsablauf zusammengefasst werden[36]. Effektiv fügt die Graphersetzungsregel zur Repräsentation dieses Modellierungsablaufs dem Graphen zwei neue Schleifen hinzu[37]. Es handelt sich hierbei entsprechend um eine *horizontal*-Kante und um eine *oneElement*-Kante.

Die Regel ist in Listing 6.6 zusätzlich zur Visualisierung als Graph mittels der *Rule and Computations Language* beschrieben. Mit Bezug auf das vorangegangene Beispiel zur Erstellung einer Linie enthält diese Regel im Mustergraphen in Zeile 3 eine bedingte Anweisung, mittels derer festgelegt wird, dass für den *Line*-Knoten *l1* nur dann eine Passung im Arbeitsgraphen gefunden werden kann, wenn das *name*-Attribut dieses Knotens den Wert *Line1* aufweist. In den Zeilen 5 und 6 wird festgelegt, dass jeweils eine neue *horizontal*- und eine neue *oneElement*-Kante erstellt werden, deren Anfangs- und Endknoten der *Line*-Knoten *l1* ist. Die *oneElement*-Kante zur Repräsentation der dimensionalen Zwangsbedingung erhält *dim1* als Identifikator, damit sie in Zeile 8 innerhalb der *eval*-Anweisung referenziert werden kann, um den Wert des Parameters festzulegen.

[36] Prinzipiell wäre es auch möglich, jeden dieser Modellierungsschritte mit einer einzelnen Graphersetzungsregel zu beschreiben.

[37] Kanten, die denselben Knoten als Start- und Endknoten haben. Dies ist hier der Fall, da sich diese Zwangsbedingungen nur auf ein geometrisches Element beziehen.

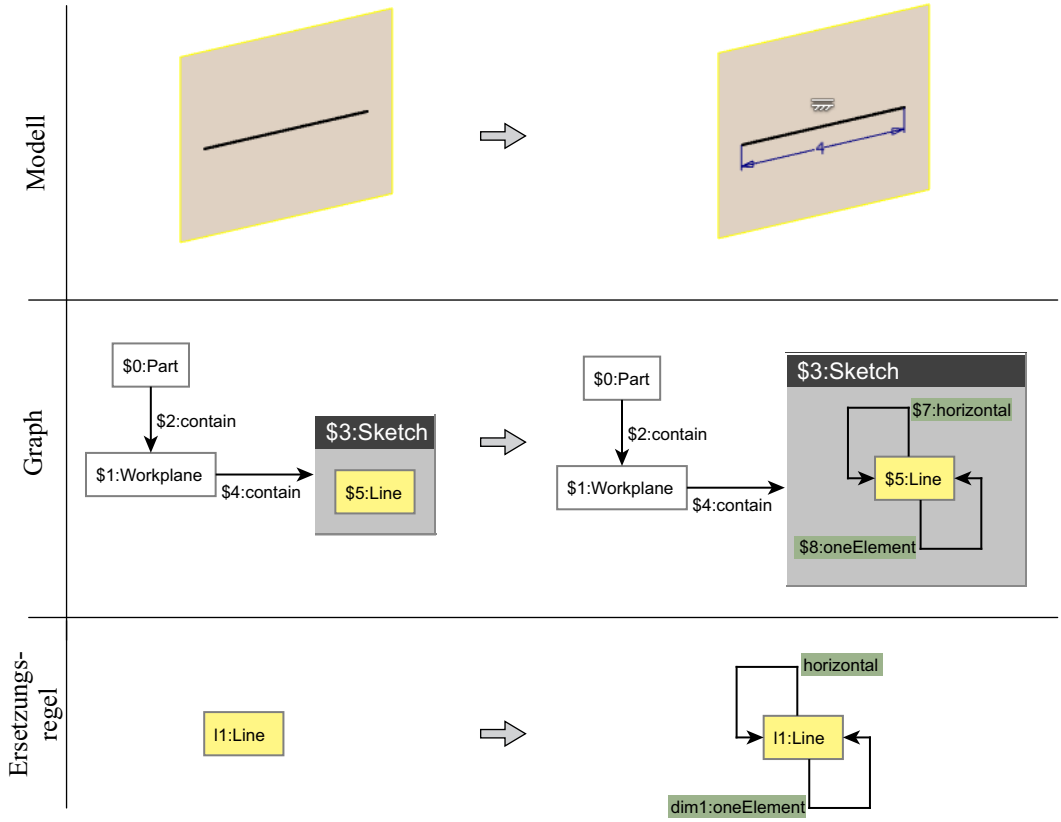


Abbildung 6.34: Im hier dargestellten Modellierungsablaufs wird eine Linie mittels einer geometrischen Zwangsbedingung als horizontal definiert. Zusätzlich wird die Länge der Linie durch eine dimensionale Zwangsbedingung auf den Wert 4 festgelegt. In der graphbasierten Repräsentation sind die Arbeitsebene und das Bauteil, zu denen die Skizze gehört, nicht dargestellt.

```
1  rule HorizontalDimension{
2      l1:Line;
3      if (l1.name == "Line1");
4  modify{
5      l1-:horizontal->l1;
6      l1-dim1:oneElement->l1;
7      eval{
8          dim1.value=4.0;
9      }
10 }}
```

Listing 6.6: Formale Definition der in Abbildung 6.34 visuell dargestellten Graphersetzungsregel mittels der *Rule and Computations Language* von GrGen.NET.

Das folgende in Abbildung 6.35 dargestellte Beispiel baut direkt auf dem Vorangegangenen auf. Hier wird auf Basis der bereits erstellten horizontalen Linie ein Quadrat erstellt. Die hierfür notwendigen geometrischen Zwangsbedingungen sind im oberen Teil der Abbildung dargestellt. Die Anordnung der Linien als Quadrat wird dabei einerseits durch die Koinzidenz

der jeweils zusammenfallenden Anfangs- bzw. Endpunkte der Linien sichergestellt. Weiterhin werden die beiden vertikalen Linien als rechtwinklig zur unteren horizontalen Linie definiert. Durch die Verwendung einer *gleich*-Zwangsbedingung wird sichergestellt, dass alle Seiten die gleiche Länge haben.

Der Mustergraph der entsprechenden Graphersetzungsregel beinhaltet einen *Sketch*- und einen *Line*-Knoten. Da diese Regel nur angewendet werden soll, wenn die entsprechende Linie horizontal ist, wird geprüft ob die y-Koordinaten von Anfangs- und Endpunkt der Linie den gleichen Wert haben (Listing 6.7, Zeile 3). Anschließend werden die neu zu erstellenden Knoten und Kanten durch den Ersetzungsgraphen definiert und die Werte der notwendigen Attribute festgelegt. Neben den Koordinaten der neu zu erstellenden Linien, die auf Basis der bestehenden Linie *l1* berechnet werden, müssen zusätzlich die *port*-Attribute der *coincident*-Kanten zur Repräsentation der Andockstellen der *koinzident*-Zwangsbedingungen mit den richtigen Werten belegt werden.

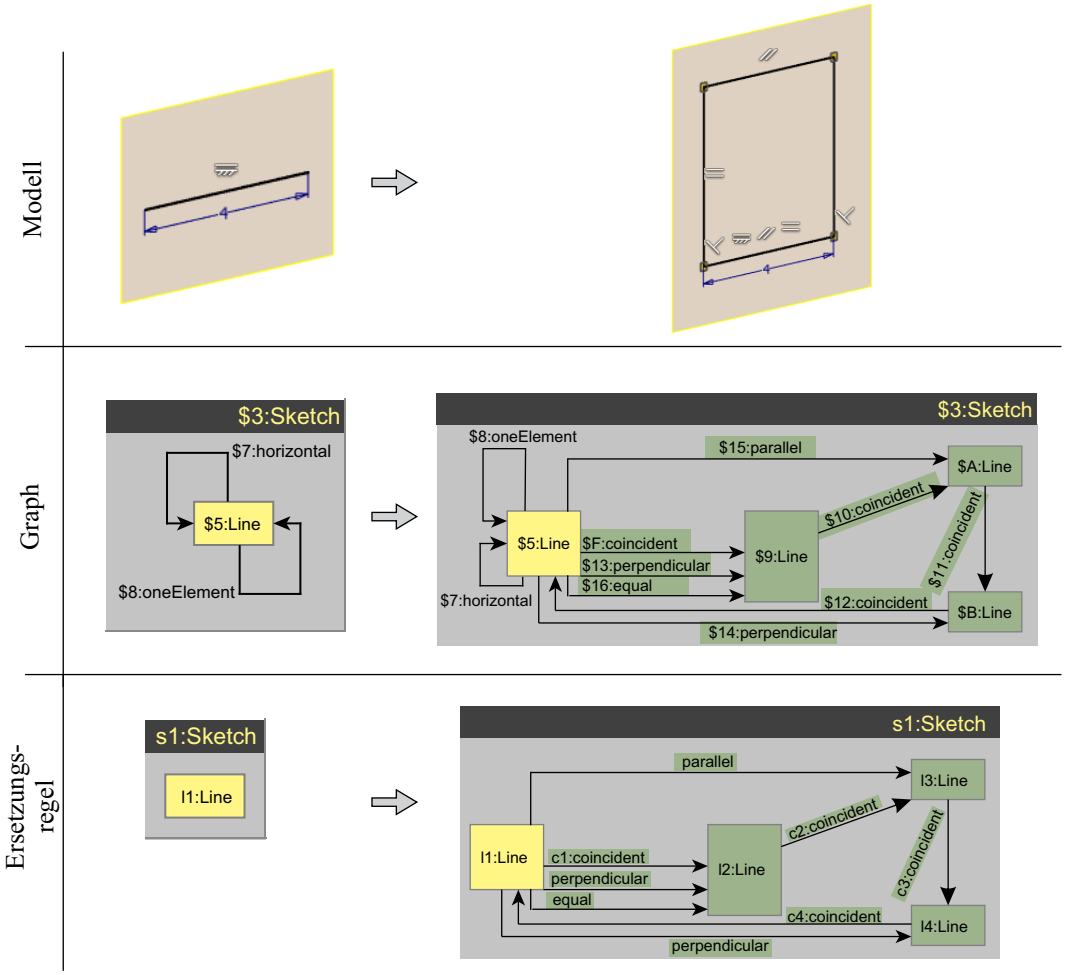


Abbildung 6.35: Im hier dargestellten Modellierungsablauf wird die Linie aus dem vorangegangenen Beispiel als Grundlage für die Erstellung eines über parametrische Zwangsbedingungen definierten Quadrats genutzt. Die dargestellte Ersetzungsregel kann auf jede horizontale Linie angewendet werden.

```
rule Square{
    s1:Sketch-:contain->l1:Line;
    if (l1.y_s == l1.y_e);
modify{
    s1-:contain->l2:Line;
    s1-:contain->l3:Line;
    s1-:contain->l4:Line;

    l1-c1:coincident->l2; l2-c2:coincident->l3;
    l3-c3:coincident->l4; l4-c4:coincident->l1;

    l1-:perpendicular->l2; l1-:perpendicular->l4;
    l1-:parallel->l3;
    l1-:equal->l2;

    eval{
        def var s:double = l1.x_e - l1.x_s;

        l1.name = "bottom"; l2.name = "left";
        l3.name = "top"; l4.name = "right";

        l2.x_s = l1.x_s;
        l2.y_s = l1.y_s;
        l2.x_e = l1.x_s;
        l2.y_e = l1.y_s + s;

        l3.x_s = l1.x_s;
        l3.y_s = l1.y_s + s;
        l3.x_e = l1.x_e;
        l3.y_e = l1.y_s + s;

        l4.x_s = l1.x_e;
        l4.y_s = l1.y_e + s;
        l4.x_e = l1.x_e;
        l4.y_e = l1.y_e;

        c1.port_s=PortType::StartPoint; c1.port_e=PortType::StartPoint;
        c2.port_s=PortType::EndPoint; c2.port_e=PortType::StartPoint;
        c3.port_s=PortType::EndPoint; c3.port_e=PortType::StartPoint;
        c4.port_s=PortType::EndPoint; c4.port_e=PortType::EndPoint;
    }
}}
```

Listing 6.7: Formale Definition der in Abbildung 6.35 visuell dargestellten Graphersetzungsregel mittels der *Rule and Computations Language* von GrGen.NET.

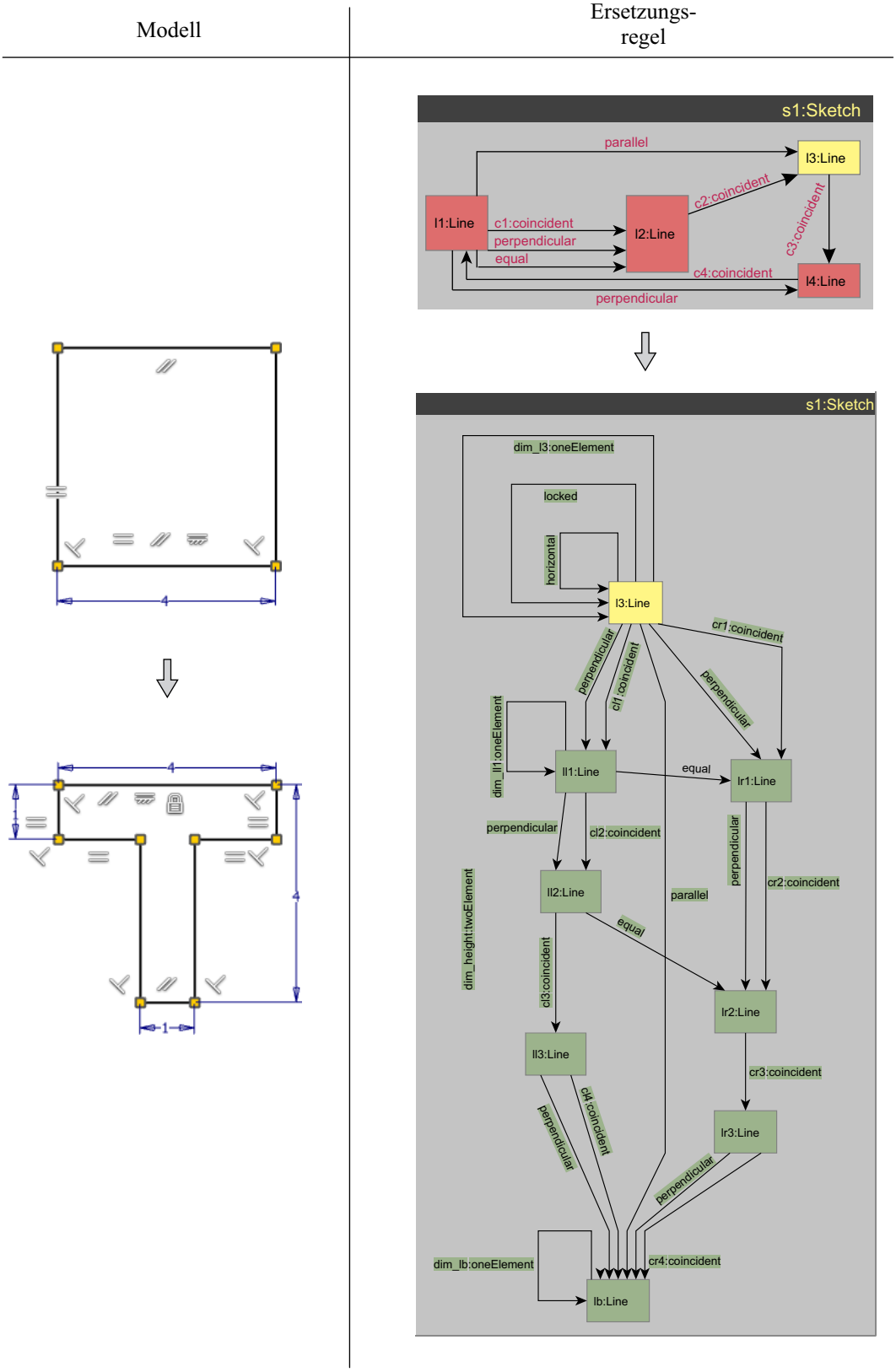


Abbildung 6.36: Grafische Visualisierung der Graphersetzungsregel zur Repräsentation des links dargestellten Modellierungsablaufs zur Detaillierung eines T-Profils.

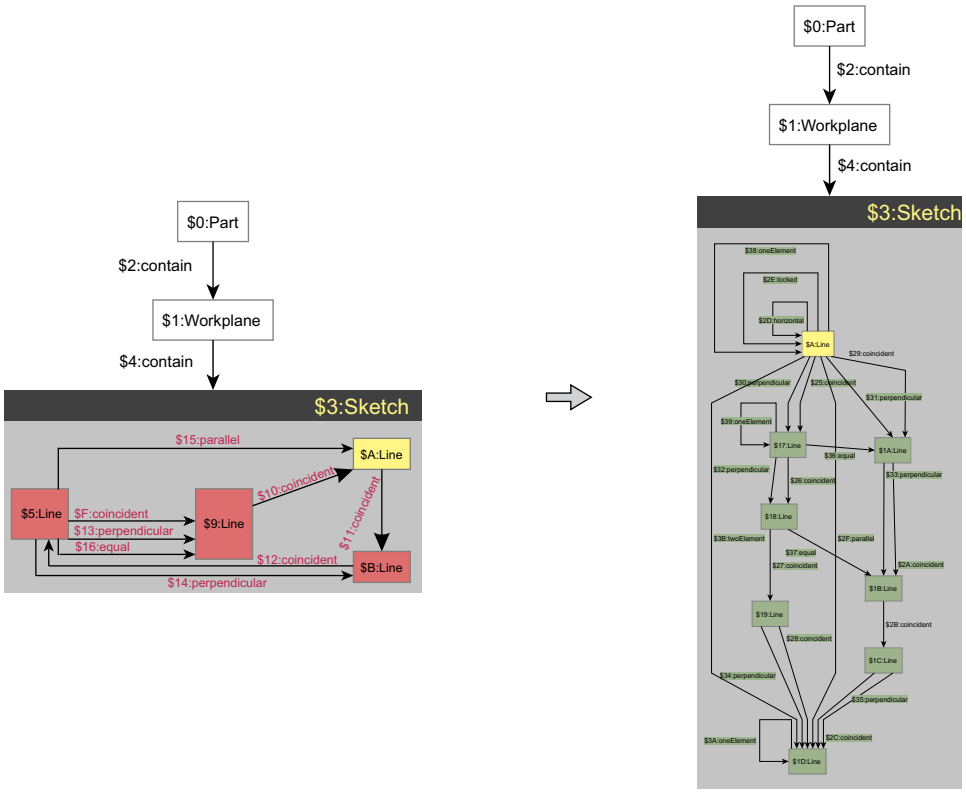


Abbildung 6.37: Anwendung der Graphersetzungsregel aus Abbildung 6.36. Links ist der Arbeitsgraph und rechts der durch die Regel modifizierte Arbeitsgraph dargestellt.

In Abbildung 6.36 ist ein drittes und deutlich komplexeres Beispiel für einen Modellierungsablauf dargestellt. Hier wird das bereits erstellte Quadrat zum Querschnitt eines T-Profils detailliert. Aus praktischer Sicht kann es sich hierbei beispielsweise um die Detaillierung eines Trägers handeln, der vor Anwendung der Regel in LOD 2 und danach in LOD 3 modelliert ist (siehe Abschnitt 2.2.2). Die durch die Regel abgeänderte Skizze beinhaltet nun ein parametrisiertes T-Profil, bei dem die Höhe und Breite, sowie die Stärke von Flansch und Steg über Parameter von dimensionalen Zwangsbedingungen gesteuert werden können. Abbildung 6.37 zeigt, wie sich die Anwendung der Regel auf den gesamten Graphen auswirkt.

Die vollständige Definition der Graphersetzungsregel ist in Listing 6.8 aufgeführt. Durch die Anwendung der Regel werden alle bestehenden Linien mit Ausnahme der oberen horizontalen Linie *l3* gelöscht. Um dieses Verhalten in GrGen.NET zu definieren, wird der Ersetzungsgraph durch die Anweisung *replace* (Zeile 6) eingeleitet. Im Gegensatz zur bisher verwendeten Anweisung *modify* werden so alle Knoten und Kanten, die nur im Mustergraphen, nicht aber im Ersetzungsgraphen referenziert sind, gelöscht. Weiterhin werden auch alle Kanten, die durch den Löschvorgang ihren Anfangs- oder Endknoten verlieren, automatisch entfernt, auch wenn dies in der Regel nicht explizit definiert ist.

Die Regel ist weiterhin so dynamisch konzipiert, dass die Höhe und Breite des Profils in Abhängigkeit der Abmessungen des Rechtecks oder Quadrats berechnet werden, auf das die Regel angewendet wird. Unabhängig davon werden die Stärken von Flansch und Steg auf den Wert von 1,0 gesetzt. Prinzipiell kann die Regel damit auf jedes zuvor erstellte Rechteck angewendet werden, sofern dessen repräsentierender Graph die im Mustergraphen definierten Bedingungen hinsichtlich der Benennung der *Line*-Knoten erfüllt (Listing 6.8, Zeile 2-5). Eine solche Einschränkung ist notwendig, damit die Regel nicht versehentlich auf vier Linien innerhalb einer Skizze angewandt wird, die kein Rechteck bilden. Auch wenn die Graphersetzung in einem solchen Fall trotzdem durchgeführt werden könnte, wäre in vielen Fällen nicht absehbar, ob dadurch ein sinnvolles Modell entsteht. Aus diesem Grund ist es bei der Definition von Regeln wichtig, den Kontext der Regel im Mustergraphen so zu definieren, dass eine fälschliche Anwendung soweit wie möglich ausgeschlossen wird. Im Fall dieser Regel wird insofern vorausgesetzt, dass der Graph auf den diese Regel angewendet wird, der gleichen Logik wie die Regel *TBeam* folgt, das heißt, dass die vier Linien aus denen sich ein Rechteck oder ein Quadrat zusammensetzt, korrekt benannt sind.

Prinzipiell wäre es aber natürlich auch möglich, den Mustergraphen so einzuschränken, dass geometrisch geprüft wird, ob vier Linien, die eine mögliche Passung des Mustergraphen im Arbeitsgraph sein könnten, tatsächlich in Form eines Rechtecks angeordnet sind. Die Abwägung, ob eine Regel eine solche Prüfung beinhaltet, hängt letztlich immer vom Szenario ab, für das die Regel konzipiert wird.

```
1  rule TBeam{
2      s1:Sketch-:contain->l1:Line; if (l1.name == "bottom");
3      s1-:contain->l2:Line; if (l2.name == "left");
4      s1-:contain->l3:Line; if (l3.name == "top");
5      s1-:contain->l4:Line; if (l4.name == "right");
6  replace{
7      s1-:contain->ll1:Line; s1-:contain->ll2:Line; s1-:contain->ll3:Line;
8      s1-:contain->lr1:Line; s1-:contain->lr2:Line; s1-:contain->lr3:Line;
9      s1-:contain->lb:Line;
10
11     l3-cl1:coincident->ll1-cl2:coincident->ll2-cl3:coincident->ll3-cl4:
           coincident->lb;
12     l3-cr1:coincident->lr1-cr2:coincident->lr2-cr3:coincident->lr3-cr4:
           coincident->lb;
13
14     l3-:horizontal->l3; l3-lock:locked->l3; l3-:parallel->lb;
15     l3-:perpendicular->ll1; l3-:perpendicular->lr1;
16     ll1-:perpendicular->ll2; lr1-:perpendicular->lr2;
17     ll3-:perpendicular->lb; lr3-:perpendicular->lb;
18
19     ll1-:equal->lr1; ll2-:equal->lr2;
20     l3-dim_l3:oneElement->l3; ll1-dim_ll1:oneElement->ll1; lb-dim_lb:
           oneElement->lb;
```

```
21      l3-dim_height:twoElement->lb;
22
23      eval{
24          def var b:double = l1.x_e - l1.x_s;
25          def var h:double = l3.y_s - l1.y_e;
26          def var flange:double = 1;
27
28          lb.x_s = l1.x_s / b; lb.y_s = l1.y_s;
29          lb.x_e = l1.x_s / b; lb.y_e = l1.y_s;
30
31          ll1.x_s = l3.x_s; ll1.y_s = l3.y_s;
32          ll1.x_e = l3.x_s; ll1.y_e = l3.y_s - flange;
33          ll2.x_s = ll1.x_e; ll2.y_s = ll1.y_e;
34          ll2.x_e = ll1.x_e / b; ll2.y_e = ll1.y_e;
35          ll3.x_s = ll2.x_e; ll3.y_s = ll2.y_e;
36          ll3.x_e = ll2.x_e; ll3.y_e = lb.x_s;
37
38          lr1.x_s = l3.x_e; lr1.y_s = l3.y_e;
39          lr1.x_e = l3.x_e; lr1.y_e = l3.y_e - flange;
40          lr2.x_s = lr1.x_e; lr2.y_s = lr1.y_e;
41          lr2.x_e = lr1.x_e / b; lr2.y_e = lr1.y_e;
42          lr3.x_s = lr2.x_e; lr3.y_s = lr2.y_e;
43          lr3.x_e = lr2.x_e; lr3.y_e = lb.x_e;
44
45          lb.name = "web_bottom"; l3.name = "flange_top";
46          ll1.name = "flange_left"; lr1.name = "flange_right";
47
48          lock.port =PortType::CenterPoint;
49          cl1.port_s=PortType::StartPoint; cl1.port_e=PortType::StartPoint;
50          cl2.port_s=PortType::EndPoint; cl2.port_e=PortType::StartPoint;
51          cl3.port_s=PortType::EndPoint; cl3.port_e=PortType::StartPoint;
52          cl4.port_s=PortType::EndPoint; cl4.port_e=PortType::StartPoint;
53          cr1.port_s=PortType::EndPoint; cr1.port_e=PortType::StartPoint;
54          cr2.port_s=PortType::EndPoint; cr2.port_e=PortType::StartPoint;
55          cr3.port_s=PortType::EndPoint; cr3.port_e=PortType::StartPoint;
56          cr4.port_s=PortType::EndPoint; cr4.port_e=PortType::EndPoint;
57
58          dim_height.port_s=PortType::Line; dim_height.port_e=PortType::Line;
59          dim_l3.value = b; dim_ll1.value = flange; dim_lb.value = 1;
                dim_height.value = h;
60      }
61  }}
```

Listing 6.8: Formale Definition der in Abbildung 6.36 visuell dargestellten Graphersetzungsregel mittels der *Rule and Computations Language* von GrGen.NET.

Repräsentation eines Modellierungsprozesses

Wie bereits zu Beginn von Abschnitt 6.5 eingeführt, umfasst ein Modellierungsprozess eine Abfolge von verschiedenen Modellierungsschritten und/oder Abläufen, die eine wesentliche konzeptionelle Weiterentwicklung des Modells im Rahmen eines modellbasierten Entwurfsprozesses beschreibt. Ein Modellierungsprozess wird daher nicht durch eine einzelne Graphersetzungsregel repräsentiert, sondern durch eine bestimmte Abfolge von Regeln, die konsekutiv ausgeführt werden und daher aufeinander abgestimmt sein müssen. Der Modellierungsprozess kann insofern als Stapelverarbeitung von Graphersetzungsregeln verstanden werden. Ein wesentlicher Nutzen besteht darin, dass so grundsätzlich ähnliche, aber in Teilen konkurrierende Abfolgen von Modellierungsoperationen formuliert werden können. Dies wird im folgenden Beispiel näher erklärt.

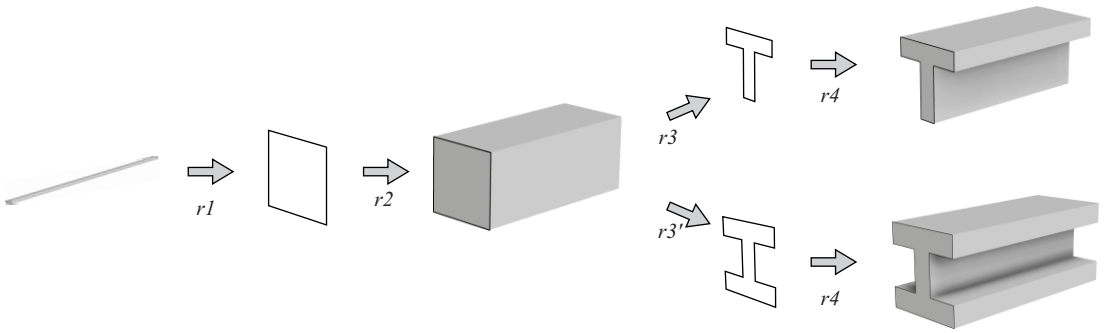


Abbildung 6.38: Zwei konkurrierende Modellierungsprozesse, die jeweils aus drei Graphersetzungsregeln zusammengesetzt sind. Auf Basis desselben Initialmodells, werden ja nach Auswahl eines Modellierungsprozesse verschiedene Querschnitt erzeugt.

Aufbauend auf den bereits eingeführten Modellierungsschritten und -abläufen sind in Abbildung 6.38 zwei Modellierungsprozesse, die sich wie folgt definieren

$$m_1 : r1 \rightarrow r2 \rightarrow r3 \rightarrow r3 \quad \text{(oben)}$$
$$m_2 : r1 \rightarrow r2 \rightarrow r3' \rightarrow r3 \quad \text{(unten)}$$

dargestellt.

Die Regeln $r1$, $r2$, $r3$ und $r4$ entsprechen dabei im Wesentlichen den Regeln *Square*, *ExtrudeFromSketch3D*, *TBeam* und *TBeam* aus den vorangegangenen Abschnitten. Die Regel $r1$ erstellt hier das Quadrat allerdings nicht auf Basis einer Linie sondern innerhalb einer leeren Skizze. Anschließend wird diese Skizze entsprechend der Länge einer Linie, welche durch eine 3D-Skizze definiert ist, extrudiert. Durch die Anwendung von $r3$ bzw. $r3'$ wird auf Basis der Skizze des Quadrats das Profil eines Trägers definiert, welches durch $r4$ wiederum extrudiert wird. Bei der Anwendung von $r3$ wird in diesem Beispiel die bestehende Skizze allerdings nicht gelöscht, sondern eine neue Skizze auf der gleichen Arbeitsebene erstellt. Im finalen Modell ist daher sowohl das extrudierte Quadrat als auch

das extrudierte Profil enthalten. Damit liegt ein mehrskaliges Modell vor, in dem ein Träger in zwei unterschiedlichen Detaillierungsgraden enthalten ist.

Dieses Beispiel zeigt, dass es sinnvoll sein kann, einen Modellierungsprozess in einzelne Schritte und Abläufe zu zerlegen, damit bestehende Regeln zur Repräsentation dieser Schritte und Abläufe in verschiedenen Prozessen wiederverwendet werden können. Zusätzlich kann durch diese Kombination von Regeln sichergestellt werden, dass die kombinierten Regeln aufeinander abgestimmt sind und so sinnvolle Resultate garantiert sind.

6.6 Technische Umsetzung

Die wesentliche Grundlage für die praktische Implementierung des beschriebenen Graphersetzungssystems bildet das Graphersetzungs-Framework GrGen.NET (siehe auch Abschnitt 4.2.6). GrGen.NET interpretiert die in der *Rule and Computations Language* formulierten Graphersetzungsregeln unter Berücksichtigung des in der *Graph Model Language* definierten Graph-Metamodells und generiert einerseits lesbaren C#-Code und kompiliert diesen zusätzlich direkt. Die so erzeugten .NET-Programmbibliotheken bilden zusammen mit den GrGen.NET-Laufzeitbibliotheken die Programmierschnittstelle (API) von GrGen.NET, die in .NET-Applikationen verwendet werden kann.

Um die durch einen Graphen repräsentierten Modelle visuell darzustellen, die parametrische Funktionalität zu testen und damit zu prüfen, ob die graphbasierte Repräsentation alle notwendigen Informationen beinhaltet, wurde die parametrische CAD-Anwendung Autodesk Inventor verwendet (siehe Abbildung 6.39). Auch Inventor bietet eine API, mittels derer die Funktionen der Software auch ohne eine Steuerung über die grafische Benutzerobefläche möglich sind. Dass auch die API von Inventor auf .NET basiert, ist ein wesentlicher Vorteil für die prototypische Implementierung.

Auf Basis der APIs von GrGen.NET und Autodesk Inventor wurde das G2I-Tool[38] prototypisch implementiert. Das G2I-Tool wurde in C# implementiert und bindet die APIs von GrGen.NET und Autodesk Inventor ein. So kann die Erstellung von Graphen entsprechend des Graph-Metamodells einerseits direkt im Code erfolgen und zusätzlich können Graphersetzungsregeln während der Laufzeit des Tools ausgeführt werden. Da die API von GrGen.NET das komplette Graph-Metamodell und die Graphersetzungsregeln als Objekte zur Verfügung stellt, kann ein während der Laufzeit erstellter Graph genutzt werden, um mittels der API von Autodesk Inventor das evaluierte Modell zu generieren. Das G2I-Tool greift hierbei auf eine laufende Instanz von Autodesk Inventor zu und ruft entsprechend der im Graphen vorhandenen Knoten und Kanten geeignete API-Funktionen auf. So wird beispielsweise ein im Graphen als Knoten repräsentiertes geometrisches Element

[38] G2I steht für *Graph to Inventor*.

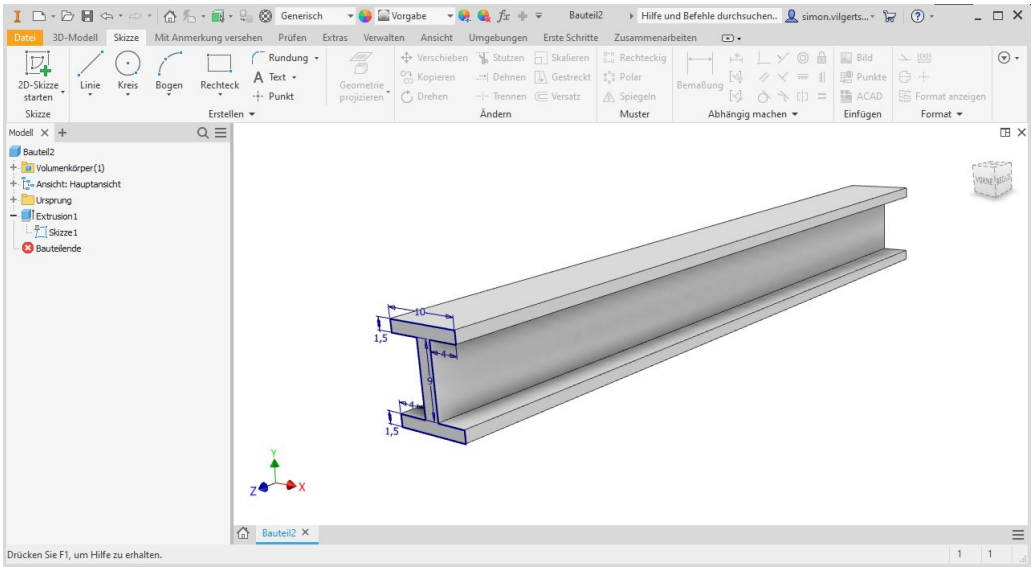


Abbildung 6.39: Visualisierung eines Modells, das auf Basis der graphbasierten Repräsentation in Autodesk Inventor automatisiert erstellt wurde.

durch den Aufruf der API-Funktion zur Erstellung dieses Elements erzeugt, wobei sowohl die Attribute des Knotens als auch der Kontext dieses Knotens im Graphen betrachtet werden. Dies ist notwendig, damit das geometrische Element auch in der Skizze erstellt wird, der es entsprechend der graphbasierten Repräsentation zugeordnet ist.

Bei der automatisierten Erstellung der geometrischen Elemente mit Hilfe von Befehlen der API von Inventor müssen natürlich die in diesem Kapitel beschriebenen Konventionen zur Interpretation des Graphen beachtet werden. Dies ist notwendig, da die graphbasierte Repräsentation kein direktes Mapping auf Funktionen der Inventor API darstellen soll, sondern den grundlegenden Definitionen der parametrischen Modellierung folgt und damit herstellerneutral bleibt. Insofern wird der Aufruf von API-Befehlen innerhalb des G2I-Tools so umgesetzt, dass diese Befehle das Modell entsprechend der durch den Graphen repräsentierten Informationen so aufbauen, dass das Resultat den in diesem Kapitel getroffenen Festlegungen entspricht.

Hier ist zu beachten, dass das durch den Graphen repräsentierte Modell und insbesondere die Skizzen in diesem Modell hinsichtlich ihrer Topologie nicht nur durch geometrische Informationen (Koordinaten) beschrieben werden, sondern auch durch parametrische Zwangsbedingungen. Durch die Verwendung der Zwangsbedingungen wird bei der Erstellung des evaluierten Modells auf die Funktionen des Geometric Constraint Solver (GCS) (siehe Abschnitt 3.2.4) zurückgegriffen. Da aber die bestehende Geometrie entsprechend Abschnitt 3.2.4 einen Einfluss auf die durch den GCS berechnete Lösung hat, ist die Definition der temporären Koordinaten im Graphen äußerst wichtig, um ein Modell korrekt zu erstellen. Die Informationen zur relativen Lage der geometrischen Elemente zueinander, die sich beim manuellen Anlegen einer Skizze automatisch ergeben, sind über die temporären

Koordinaten in den Graphen integriert. Diese legen für jedes geometrische Element fest, an welcher Position es initial platziert wird. Sie können sich dabei auf die Koordinaten bereits bestehender geometrischer Elemente beziehen (dies kann innerhalb einer Graphersetzungsregel definiert werden), sodass die neuen Elemente relativ zu diesen positioniert werden. Diese Koordinaten werden „temporär“ genannt, da sie vom GCS nach erfolgreicher Lösung des Systems der Zwangsbedingungen gegebenenfalls überschrieben werden (siehe auch Abschnitt 5.5.1 und Abschnitt 6.2.5).

Durch diese zusätzlichen Informationen können nun die API-Befehle im G2I-Tool so angepasst werden, dass die geometrischen Elemente schon bei der Erstellung annähernd an ihren finalen Positionen eingefügt werden, ohne sie jedoch dabei dort fest zu verankern. Dies geschieht erst anschließend durch das Hinzufügen der parametrischen Zwangsbedingungen. Somit ist einerseits die richtige Topologie der Skizze und die richtige relative Lage der Elemente untereinander festgelegt und andererseits können Abmessungen und Abstände weiterhin durch das Ändern von Parametern angepasst werden.

In Slepicka (2019) wurde ergänzend untersucht, ob auch in den parametrischen CAD-Anwendungen Siemens NX und FreeCAD eine Erzeugung von Modellen auf Basis der graphbasierten Repräsentation umsetzbar ist. Hierzu wurde für beide Anwendungen eine Variante des G2I-Tools entwickelt, die auch die API von GrGen.NET verwenden, aber anstelle der API von Inventor auf die API von Siemens NX bzw. FreeCAD zugreifen. In der Arbeit von Slepicka konnte gezeigt werden, dass die entwickelte graphbasierte Repräsentation parametrischer Modelle ausreichend allgemein ist, um sie als Basis für die Erstellung von Modellen in unterschiedlichen CAD-Anwendungen zu nutzen.

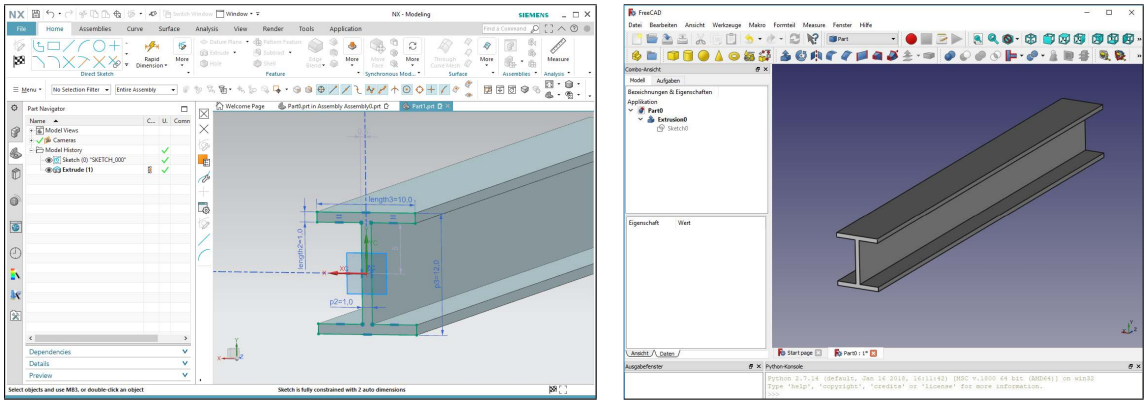


Abbildung 6.40: Auf Basis von Graphen, die mit dem hier vorgestellten Graphersetzungssystem generiert werden, können grundsätzlich auch Modelle in den parametrischen CAD-Anwendungen Siemens NX (links) und FreeCAD (rechts) erzeugt werden. Hierzu ist jedoch genau wie für Autodesk Inventor eine entsprechende Implementierung notwendig, die einen Graphen interpretiert und durch die Verwendung der jeweiligen API ein parametrisches Modell generiert. *Quelle: Slepicka, 2019.*

Hinsichtlich der technischen Umsetzung und prototypischen Implementierung ist schlussendlich festzuhalten, dass die APIs der gängigen parametrischen CAD-Anwendungen,

wie bereits in Abschnitt 2.1.3, Seite 22 angemerkt, nicht in erster Linie darauf ausgelegt sind, Modelle auf Basis von externen Daten zu erstellen bzw. anwendungsübergreifenden Konventionen zum Aufbau parametrischer Modelle zu entsprechen. Die APIs folgen vielmehr der Logik und dem Aufbau der jeweiligen CAD-Anwendung. Dies hat zur Folge, dass bei der Verarbeitung der graphbasierten Repräsentation neben den hier erfolgten Definitionen auch immer berücksichtigt werden muss, wie einzelne Aspekte der parametrischen Modellierung in der jeweiligen CAD-Anwendung umgesetzt werden. Dies ist letztlich nur für solche parametrischen Funktionalitäten möglich, die in allen betrachteten Anwendungen nach den gleichen grundlegenden Prinzipien umgesetzt werden und erfordert weiterhin eine umfangreiche Einarbeitung in die API der jeweiligen Anwendung.

6.7 Zusammenfassung

In diesem Kapitel wurde die Entwicklung eines Graphersetzungssystem zur Beschreibung von Modellierungsoperationen zur Erstellung von parametrischen Modellen detailliert beschrieben.

Hierbei wurde dokumentiert, wie das Graphersetzungssystem aufgebaut ist und wie die damit erzeugten Graphen interpretiert werden müssen, um ein evaluiertes parametrisches Modell abzuleiten. Neben der formalen Definition des Graphersetzungssystems wurden die verschiedenen Typen von Knoten und Kanten sowie die zugehörigen Attribute hinsichtlich ihrer Funktionen beschrieben und in Form eines Graph-Metamodells definiert. Auf dieser Basis wurde anhand verschiedener Beispiele erläutert, wie einzelne Graphen ein bestimmtes parametrisches Modell repräsentieren und anschließend dargestellt, wie Veränderungen des Modells in Form von Modellierungsschritten, -abläufen und -prozessen beschrieben werden können.

Kapitel 7

Fallstudien

In diesem Kapitel wird anhand von drei Fallstudien beispielhaft gezeigt, wie Modellierungsschritte in einem praktischen Kontext durch Graphersetzungsregeln ausgedrückt werden können. Die Beispiele basieren dabei auf dem im vorangegangenen Kapitel beschriebenen Graph-Metamodell und den Konventionen zur Interpretation der darauf basierenden Graphen.

Zur Umsetzung der folgenden Beispiele ist dabei keine Erweiterung des Graph-Metamodells notwendig. Vielmehr wird das Graphersetzungssystem um zusätzliche Graphersetzungsregeln erweitert. Durch diese Graphersetzungsregeln werden Modellierungsoperation in verschiedenen Kontexten formalisiert. So lässt sich zeigen, dass die Möglichkeit zur Erweiterung des Graphersetzungssystems prinzipiell gegeben ist und der vorgestellte Ansatz grundsätzlich in verschiedenen Bereichen angewendet werden kann.

In den folgenden Abschnitten werden dazu Szenarien, in denen ein Graphersetzungssystem im Kontext der parametrischen Modellierung eingesetzt werden kann, eingeführt und die zugehörige Erweiterung des Graphersetzungssystems durch entsprechende Graphersetzungsregeln beschrieben.

7.1 Modellierung eines Tunnels in verschiedenen Detaillierungsgraden

In diesem Anwendungsbeispiel werden die in Abschnitt 2.2.3 beschriebenen Vorarbeiten von Borrmann *et al.* (2014), Borrmann *et al.* (2013a), Borrmann *et al.* (2015) aufgegriffen. Das dort vorgestellte Konzept zur Erstellung mehrskaliger Modelle von Schildvortriebstunneln wurde in der Dissertation von Jubierre (2016) detailliert ausgearbeitet und bildet die Grundlage für die im Folgenden beschriebenen Modellierungsoperationen und deren graphbasierte Formalisierung. Hintergrund der Arbeiten von Jubierre und Borrmann *et al.* sind die Herausforderungen, die sich während der modellbasierten Planung von Infrastrukturbauwerken durch die zu betrachtenden stark variierenden Größenordnungen ergeben. Insbesondere bei Tunnelbauwerken müssen sowohl die Trasse des Bauwerks im Kilometerbereich als auch Details der technischen Ausstattung oder Querschläge und Rettungsschächte im Zentimeterbereich betrachtet werden (Borrmann *et al.*, 2015).

Während diese Problematik in der konventionellen zeichnungsbasierten Planung durch die Verwendung unterschiedlicher Maßstäbe gelöst wird, werden dreidimensionale Bauwerksmodelle im Maßstab 1:1 modelliert (siehe auch Abschnitt 2.2.2). Bezugnehmend auf Vorarbeiten im GIS-Bereich (Sester, 2001) führen Borrmann *et al.* (2013a) daher fünf Detaillierungsgrade für Tunnelbauwerke ein. In Borrmann *et al.* (2014) wird weiterhin beschrieben, wie das mehrskalige Modell, auch über die verschiedenen Detaillierungsgrade hinweg, durch parametrische Modellierung konsistent gehalten werden kann. Im vorgestellten Ansatz werden so Änderungen, die in einem niedrigen Detaillierungsgrad vorgenommen wurden, automatisch auf höhere Detaillierungsgrade propagiert, indem die geometrischen Elemente in verschiedenen Detaillierungsgraden durch parametrische Zwangsbedingungen miteinander verknüpft werden. Dies wird in den LoDs 1-4 durch die Definition einer LoD-übergreifenden räumlichen Struktur des Tunnelbauwerks erreicht. Die Autoren stellen allerdings weiterhin fest, dass die Erstellung der entsprechenden parametrischen Modelle mit einem hohen manuellen Aufwand verbunden ist und der Entwurfsprozess daher durch Ansätze zur Automatisierung von Modellierungsoperationen unterstützt werden sollte.

Abbildung 7.1 zeigt vier der fünf Detaillierungsgrade, die im Rahmen der genannten Arbeiten von Borrmann *et al.* definiert wurden und hier weiter betrachtet werden. Hier werden ausgehend von der Trasse *(TunnelAxis)* des Bauwerks im LoD 1[1] als Kurve im Raum, die höheren LoDs als Volumenkörper zur Beschreibung der räumlichen Struktur des Tunnelbauwerks erstellt. Bei der parametrischen Modellierung muss also beachtet werden, dass die Änderung des Trassenverlaufs in LoD 1 auf die Geometrie des *FullTunnelSpace* im nächsthöheren LoD propagiert wird. Entsprechend muss eine Änderung[2] des *FullTunnel-*

[1] Im Folgenden wird die von Borrmann *et al.* verwendete Schreibweise LoD als Abkürzung für den Detaillierungsgrad verwendet.

[2] Beispielsweise eine Änderung des Durchmessers.

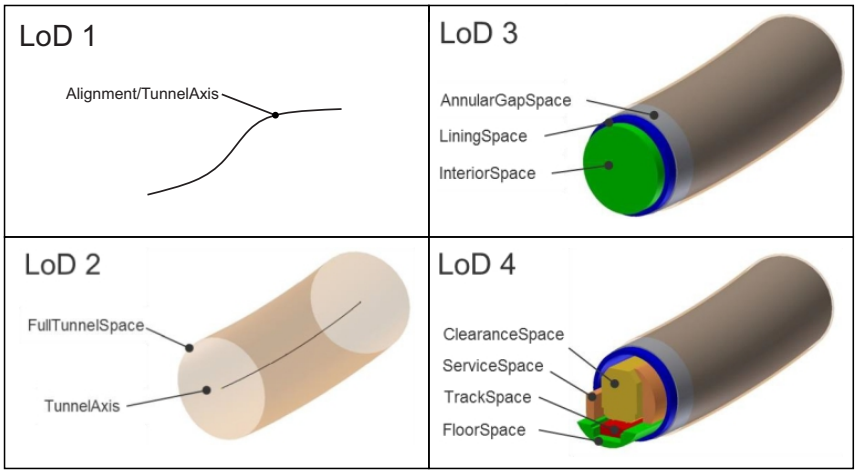


Abbildung 7.1: Detaillierungsgrade zur mehrskaligen Modellierung eines Schildtunnels. *Die Abbildungen der LoD 2-4 wurden aus Borrmann* et al. *(2013a) übernommen.*

Space im LoD 2 auf die in LoD 3 definierten räumlichen Bereiche übertragen werden. Dies gilt genauso für die räumlichen Bereiche des LoD 4, die vollständig im *InteriorSpace* des LoD 3 enthalten sind. Im Folgenden werden die Modellierungsoperationen beschrieben, die zur Erstellung dieses mehrskaligen Modells notwendig sind und die Repräsentation dieser Modellierungsoperationen durch Graphersetzungsregeln vorgestellt. Vom Prinzip her basieren diese Modellierungsoperationen auf den Beschreibungen zur Erstellung parametrischer Modellen zur automatischen Konsistenzsicherung, die von Jubierre (2016) erarbeitet und im Rahmen eines Fallbeispiels (Neubau der 2. S-Bahn-Stammstrecke München) angewandt wurden.

LoD 1

Die in Abbildung 7.2 dargestellte erste Modellierungsoperation besteht in der Erstellung einer 3D-Skizze im Raum, durch die der Verlauf der Trasse beschrieben wird. Voraussetzung für diesen Modellierungsablauf ist, dass bereits ein leeres Bauteil erstellt wurde. Die entsprechende Graphersetzungsregel *LoD0* ist in Listing 7.1 dargestellt. Der Mustergraph besteht hier lediglich in einem *Part*-Knoten, dem im Ersetzungsgraph ein *Sketch3D*-Knoten und drei *Point3D*-Knoten hinzugefügt werden. In diesem Beispiel wird der Verlauf der Trasse nur durch drei Punkte beschrieben[3], um den Graph übersichtlich zu halten. Die Zugehörigkeit der 3D-Skizze zum Bauteil und der Punkt zur 3D-Skizze wird über *contain*-Kanten festgelegt. Um die 3D-Skizze, die den Trassenverlauf beschreibt, in Mustergraphen weiterer Regeln eindeutig definieren zu können, wird das *name*-Attribut dieses Knotens mit dem Wert *TunnelAxis* belegt.

[3]Grundsätzlich könnte diese Regel aber auch eine große Anzahl von Punkten erzeugen, und so einen Trassenverlauf detailliert beschreiben.

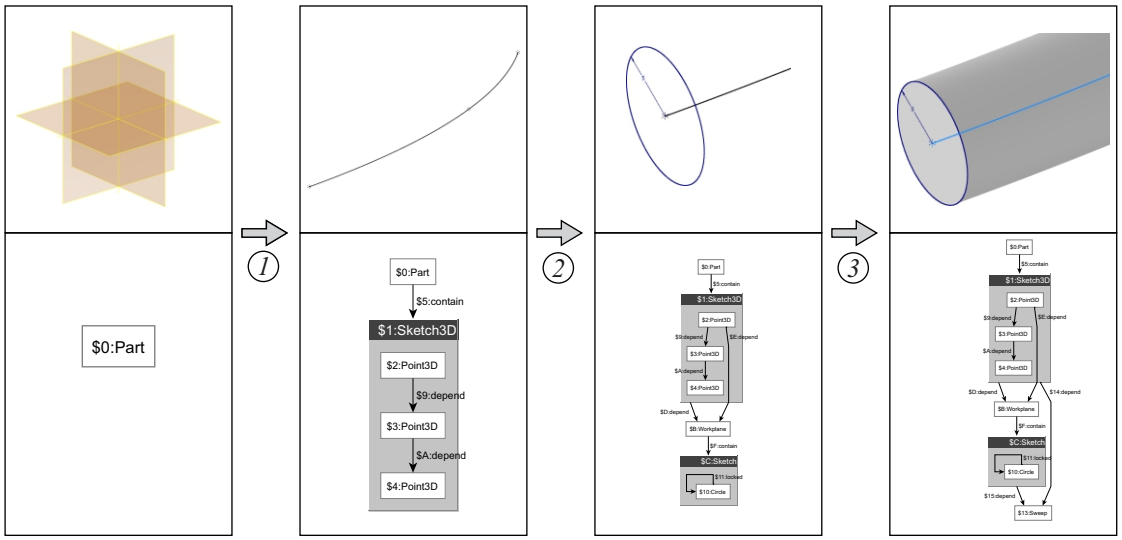


Abbildung 7.2: Detaillierung des Modells eines Schildvortriebstunnels. Schritt 1 erstellt die Trasse des Bauwerks und den LoD 1. Mit den Schritten 2 und 3 wird der LoD 2 erstellt. Die Geometrie des LoD 2 ist dabei mittels parametrischer Zwangsbedingungen vom LoD 1 und damit vom Verlauf der Trasse abhängig.

LoD 2

Auf Basis der durch die 3D-Skizze definierten Trasse kann nun eine Arbeitsebene erstellt, werden, die die Grundlage für eine Skizze bildet. In dieser Skizze wird anschließend ein Kreis zur Beschreibung des *FullTunnelSpace* modelliert. Dies entspricht Schritt 2 in Abbildung 7.2 und wird durch die Ersetzungsregeln *LoD2_1* und *LoD2_2* in Listing 7.2 repräsentiert, wobei mittels *LoD2_1* die Arbeitsebene sowie eine leere Skizze und mittels *LoD2_2* der Kreis erstellt wird. Der Kreis wird dabei mittels einer *fixiert*-Zwangsbedingung festgehalten. Dadurch verschiebt sich der Kreis bei einer durch die Änderung der 3D-Skizze initiierten Verschiebung von Arbeitsebene und Skizze automatisch so, dass der Mittelpunkt des Kreises weiterhin auf dem ersten Punkt der 3D-Skizze liegt.

Anschließend wird der dritte in Abbildung 7.2 dargestellte Schritt mittels der in Listing 7.3 definierten Regel durchgeführt. So wird auf Basis der zuvor erstellten Skizze – bzw. dem in der Skizze modellierten Kreis – eine Sweeping Operation entlang der *TunnelAxis* durchgeführt. Damit wird der Volumenkörper, der den *FullTunnelSpace* beschreibt, erstellt und der LoD 2 des Tunnels vollständig parametrisch modelliert.

```
rule LoD0{
    p:Part;
modify{
    p-:contain->s:Sketch3D; s-:contain->p1:Point3D;
    s-:contain->p2:Point3D; s-:contain->p3:Point3D;
    p1-:depend->p2-:depend->p3;
eval{
    s.type = PathType::interpolate; s.name = "TunnelAxis";
    p1.x = 0.0; p1.y = 0.0; p1.z = 0.0;
    p2.x = 60.0; p2.y = -5.0; p2.z = 0.0;
    p3.x = 0.0; p3.y = 0.0; p3.z = 0.0;
    }
}}
```

Listing 7.1: Die hier definierte Graphersetzungsregel führt den in Abbildung 7.2 mit 1 gekennzeichneten Modellierungsablauf durch.

```
rule LoD2_1{
    s3d:Sketch3D; p:Point3D; s3d-:contain->p;
    negative {:Point3D-:depend->p;}
modify{
    wp:Workplane; s3d-:depend->wp;
    p-:depend->wp;
    wp-:contain->s:Sketch;
eval{
    wp.type = WpType::context;
    s.name = "FullTunnelSpaceSketch";
    }
}}
rule LoD2_2{
    s:Sketch; if (s.name == "FullTunnelSpaceSketch");
modify{
    c:Circle;
    c-:locked->c; s-:contain->c;
    c-dim:oneElement->c;
eval{
    c.name = "FullTunnelSpaceCircle";
    c.x = 0.0; c.y = 0.0; c.r = 5.0;
    dim.value = c.r;
    }
}}
```

Listing 7.2: Die hier definierten Graphersetzungsregeln führen die in Abbildung 7.2 mit 2 gekennzeichnete Modellierungsoperation durch. Mittels der Regel *LoD2_1* wird zuerst eine Arbeitsebene und eine leere Skizze erstellt. Die Arbeitsebene wird dabei so definiert, dass sie durch den ersten Punkt der 3D-Skizze verläuft und orthogonal zum Verlauf der in dieser 3D-Skizze definierten Trasse ist. *LoD2_2* erstellt einen Kreis in dieser Skizze.

```
rule LoD2_3{
    s3d:Sketch3D; if (s3d.name == "TunnelAxis");
    s:Sketch; if (s.name == "FullTunnelSpaceSketch");
modify{
    sw:Sweep; s3d-:depend->sw; s-:depend->sw;
eval{
    sw.name = "FullTunnelSpace";}
}}
```

Listing 7.3: Graphersetzungsregel, die einen Sweep der Skizze des *FullTunnelSpace* entlang der *TunnelAxis* erzeugt. Dies entspricht der dritten Modellierungsoperation, die in Abbildung 7.2 dargestellt ist und vervollständigt die Modellierung des Volumenkörpers zur Beschreibung des *FullTunnelSpace*.

LoD 3

Um den LoD 3 zu erstellen, müssen drei neue Skizzen als Grundlage für den Sweep von *AnnularGapSpace*, *LiningSpace* und *InteriorSpace* erstellt werden. Dies ist in Abbildung 7.3 durch die Schritte 4, 5 und 6 dargestellt. Die Skizzen für den *AnnularGapSpace* und den *LiningSpace* bestehen dabei jeweils aus zwei konzentrischen Kreisen, die je einen Kreisring bilden. In beiden Skizzen wird der äußere Kreis aus einer bereits vorhandenen Skizze projiziert. Damit wird sichergestellt, dass die äußere Begrenzung des *AnnularGapSpace* vom *FullTunnelSpace* und die äußere Begrenzung des *LiningSpace* wiederum von der inneren Begrenzung des *LiningSpace* abhängig sind. Die Skizze, die die Grundlage für den *InteriorSpace* bildet, beinhaltet nur einen Kreis, der vom inneren Kreis des *LiningSpace* projiziert wird. Eine Änderung der Trasse oder des Durchmessers des *FullTunnelSpace* wird damit automatisch in den LoD 3 übertragen. Die Wanddicken von *AnnularGapSpace* und *LiningSpace* werden mittels dimensionaler Zwangsbedingungen definiert. Eine Änderung der Parameterwerte durch die Verwendung der projizierten Geometrie kann so automatisch auf die jeweils weiter innen liegende Geometrie übertragen werden. Die Graphersetzungsregeln *LoD3_AGS* , *LoD3_LS* und *LoD3_IS*, mit denen diese Modellierungsabläufe ausgeführt werden können, sind in Listing 7.4 aufgeführt. Die Regeln *LoD3_AGS* und *LoD3_LS* sind dabei vom Aufbau weitgehend identisch und unterscheiden sich nur durch einzelne Attributwerte.

Nachdem diese drei Skizzen erstellt wurden, muss im Anschluss lediglich eine Regel ausgeführt werden, die die entsprechenden Volumenkörper zur Beschreibung von *AnnularGapSpace*, *LiningSpace* und *InteriorSpace* durch Sweeping Operationen erstellt. Diese Regel entspricht vom grundsätzlichen Aufbau der bereits für den LoD 2 definierten Regel, wobei in diesem Fall durch eine Regel drei Sweeping Operationen ausgeführt werden. Diese Regel ist in Anhang B enthalten.

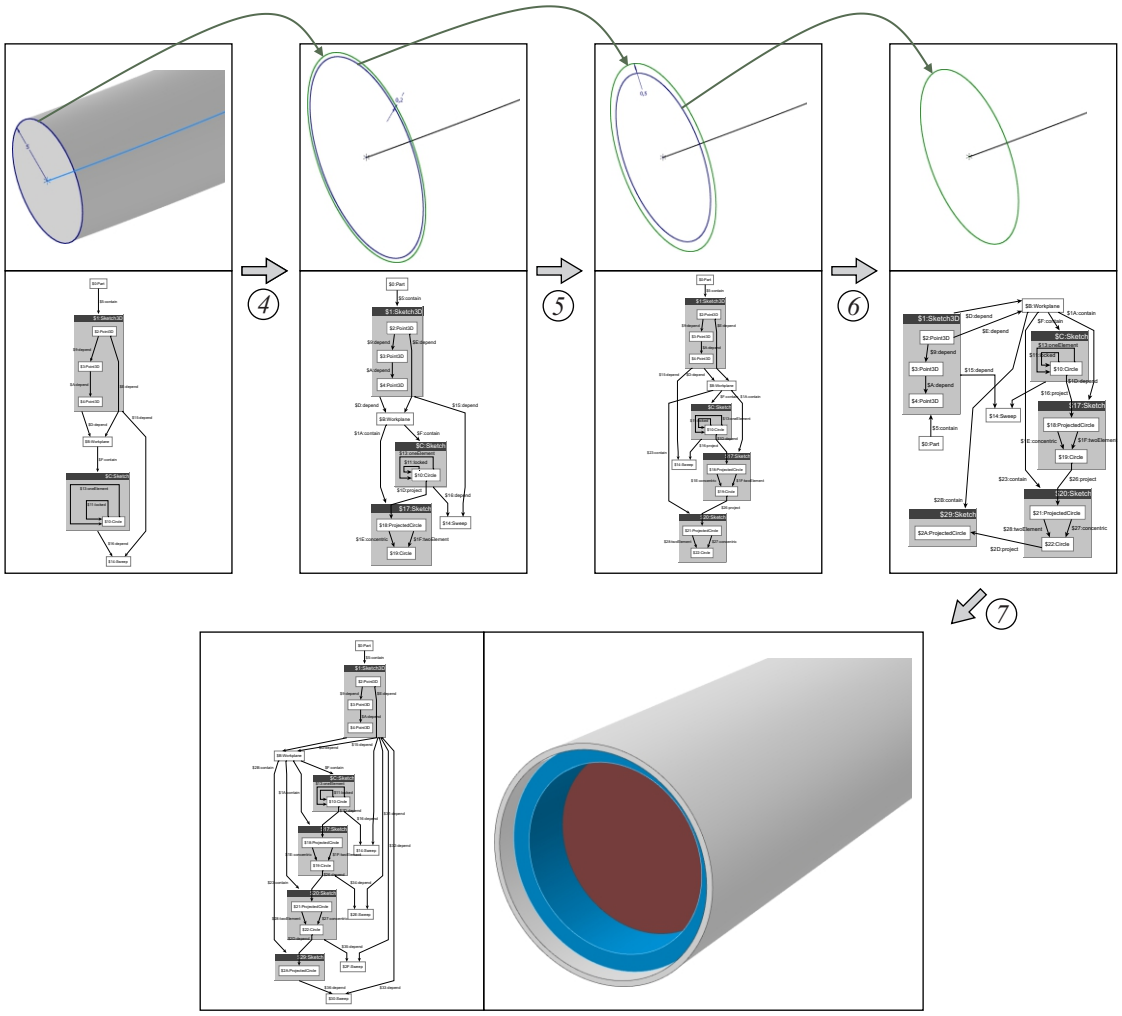


Abbildung 7.3: Weitere Detaillierung des Modelles eines Schildvortriebstunnels. Mit den Schritten 4, 5 und 6 wird die Geometrie des LoD 3 als Basis für die Sweeping Operation in Schritt 6 erstellt. Die grünen Pfeile kennzeichnen, dass die Kreise jeweils in weitere Skizzen projiziert werden, um letztlich die gesamte Geometrie vom *FullTunnelSpace* und damit vom Trassenverlauf abhängig zu machen.

```
1  rule LoD3_AGS{
2      wp:Workplane-:contain->s:Sketch-:contain->c:Circle;
3      if (s.name == "FullTunnelSpaceSketch");
4  modify{
5      s1:Sketch;
6      pc:ProjectedCircle; c1:Circle;
7      wp-:contain->s1; s1-:contain->pc; s1-:contain->c1;
8      c-:project->pc;
9      pc-:concentric->c1; pc-dim:twoElement->c1;
10 eval{
11     s1.name = "AnnularGapSpaceSketch";
12     dim.value = 0.2; dim.mode = ModeType::Direct;
13     dim.port_s = PortType::Arc; dim.port_s = PortType::Arc;
```

```
14      c1.x = c.x; c1.y = c.y; c1.r = c.r * 0.1;
15      }
16  }}
17  rule LoD3_LS{
18      wp:Workplane-:contain->s:Sketch-:contain->c:Circle;
19      if (s.name == "AnnularGapSpaceSketch");
20  modify{
21      s1:Sketch;
22      pc:ProjectedCircle; c1:Circle;
23      wp-:contain->s1; s1-:contain->pc; s1-:contain->c1;
24      c-:project->pc;
25      pc-:concentric->c1; pc-dim:twoElement->c1;
26  eval{
27      s1.name = "LiningSpaceSketch";
28      dim.value = 0.5; dim.mode = ModeType::Direct;
29      dim.port_s = PortType::Arc; dim.port_s = PortType::Arc;
30      c1.x = c.x; c1.y = c.y; c1.r = c.r * 0.1;
31      }
32  }}
33  rule LoD3_IS{
34      wp:Workplane-:contain->s:Sketch-:contain->c:Circle;
35      if (s.name == "LiningSpaceSketch");
36  modify{
37      s1:Sketch;
38      pc:ProjectedCircle;
39      wp-:contain->s1; s1-:contain->pc;
40      c-:project->pc;
41  eval{
42      s1.name = "InteriorSpaceSketch";
43      }
44  }}
```

Listing 7.4: Die hier definierten Graphersetzungsregeln *Lod3_ASG*, *Lod3_LS* und *Lod3_IS* bilden zusammen einen Modellierungsablauf, der die für den LoD 3 notwendigen Skizzen zur Modellierung des *AnnularGapSpace*, *LiningSpace* und *InteriorSpace* erstellt. Die entsprechenden Modellierungsabläufe sind visuell in Abbildung 7.3 dargestellt.

LoD 4

Da im LoD 4 nur der *InteriorSpace* aus LoD 3 weiter detailliert wird, bildet der entsprechende Kreis die wesentliche Grundlage für die folgenden Modellierungsoperationen. Zuerst wird in einem Modellierungsprozess der *ClearanceSpace* erstellt. Die entsprechenden Modellierungsabläufe sind in Abbildung 7.4 dargestellt.

Im ersten Ablauf (Regel *LoD4_ClearanceSpace_1*) wird der Kreis, der den *InteriorSpace* abgrenzt, in eine neue Skizze projiziert und eine horizontale Linie mit einem bestimmten

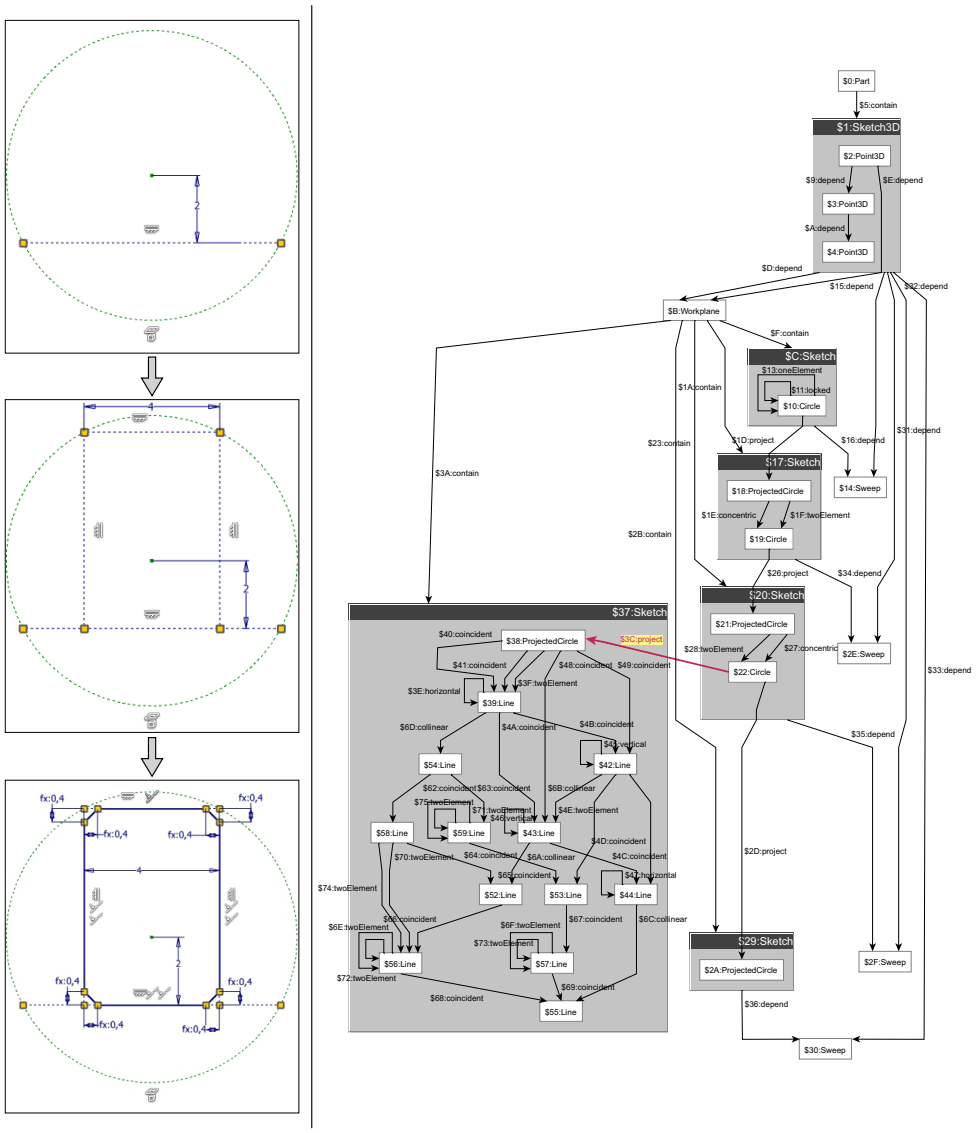


Abbildung 7.4: Modellierungsabläufe zur Erstellung des *ClearanceSpace* in LoD 4 (links) und der durch die Anwendung der entsprechenden Regeln erstellte Graph (rechts).

Abstand vom Mittelpunkt des Kreises als Konstruktionsgeometrie erstellt. Start- und Endpunkt der Linie sind koinzident mit dem Kreisbogen. Diese Linie dient im Folgenden als horizontale Abgrenzung, an der sich alle *Spaces* im LoD 4 orientieren. Der nächste Ablauf (Regel *LoD4_ClearanceSpace_2*) erstellt drei weitere Linien als Konstruktionsgeometrie, die den oberen Teil des *InteriorSpace* zwischen *ClearanceSpace* und den *ServiceSpaces* aufteilen. Das so entstandene Rechteck bildet die Begrenzung des *ClearanceSpace*, dessen Breite über eine dimensionale Zwangsbedingung gesteuert werden kann. Die Höhe ergibt sich durch die parametrischen Zusammenhänge automatisch aus dem Radius des Kreises und der Position der horizontalen Linie. Der dritte Ablauf (Regel *LoD4_ClearanceSpace_1*) erstellt basierend auf der bereits existenten Konstruktionsgeometrie ein Rechteck mit abgefasten Ecken, das das Profil des *ClearanceSpace* definiert. Allein in diesem dritten Modellierungsablauf wird

eine Vielzahl von parametrischen Zwangsbedingungen erstellt, um die entsprechende Skizze voll zu bestimmen (u.a. acht *koinzident*- und acht dimensionale Zwangsbedingungen, bei denen jeweils die *port*-Attribute definiert werden müssen).

Hierbei ist zu beachten, dass die dimensionalen Zwangsbedingungen, die die Abmessung der Fase definieren, in diesem Beispiel jeweils von der Breite des *ClearanceSpace* abhängen. Grundsätzlich könnte hier natürlich auch ein fester Wert in der Graphersetzungsregel festgelegt oder der Wert der jeweiligen Parameter nachträglich im Modell angepasst werden. Dass die komplette Geometrie des *ClearanceSpace* vom *InteriorSpace* abhängig ist, zeigt auch die Visualisierung des Graphen in Abbildung 7.4. Die entsprechende Kante, die die Projektion des Kreises repräsentiert, ist hier in rot hervorgehoben.

```
1  rule LoD4_ClearanceSpace_3{
2      s:Sketch; if (s.name == "ClearanceSpaceSketch");
3      s-:contain->l_b:Line; if (l_b.name == "L_bottom");
4      s-:contain->l_t:Line; if (l_t.name == "L_top");
5      s-:contain->l_l:Line; if (l_l.name == "L_left");
6      s-:contain->l_r:Line; if (l_r.name == "L_right");
7  modify{
8      l:Line; r:Line; s-:contain->l; s-:contain->r;
9      b:Line; t:Line; s-:contain->b; s-:contain->t;
10     lt:Line; rt:Line; s-:contain->lt; s-:contain->rt;
11     bl:Line; br:Line; s-:contain->bl; s-:contain->br;
12     b-co1:coincident->bl; b-co2:coincident->br;
13     bl-co3:coincident->l; br-co4:coincident->r;
14     l-co5:coincident->lt; r-co6:coincident->rt;
15     lt-co7:coincident->t; rt-co8:coincident->t;
16     l_l-:collinear->l; l_r-:collinear->r;
17     l_t-:collinear->t; l_b-:collinear->b;
18     lt-dim_lt_h:twoElement->lt; rt-dim_rt_h:twoElement->rt;
19     bl-dim_bl_h:twoElement->lt; br-dim_br_h:twoElement->br;
20     lt-dim_lt_v:twoElement->lt; rt-dim_rt_v:twoElement->rt;
21     bl-dim_bl_v:twoElement->lt; br-dim_br_v:twoElement->br;
22 eval{
23     //siehe vollständige Definition der Regel in Anhang B
24     }
25 }}
```

Listing 7.5: Graphersetzungsregel zur Erstellung des Rechtecks mit abgefasten Ecken, das das Profil des *ClearanceSpace* definiert. Der Teil der Regel, in dem die Attributwerte der Knoten und Kanten festgelegt werden ist hier nicht enthalten. Die vollständige Regel ist in Anhang B enthalten.

Die Graphersetzungsregel, die den dritten Modellierungsablauf im Modellierungsprozess zur Erstellung des *ClearanceSpace* repräsentiert, ist teilweise in Listing 7.5 definiert. Es wurde dabei auf die Darstellung des Teils der Ersetzungsregel, in dem unter anderem die Werte der temporären Koordinaten und der *port*-Attribute definiert werden, zugunsten

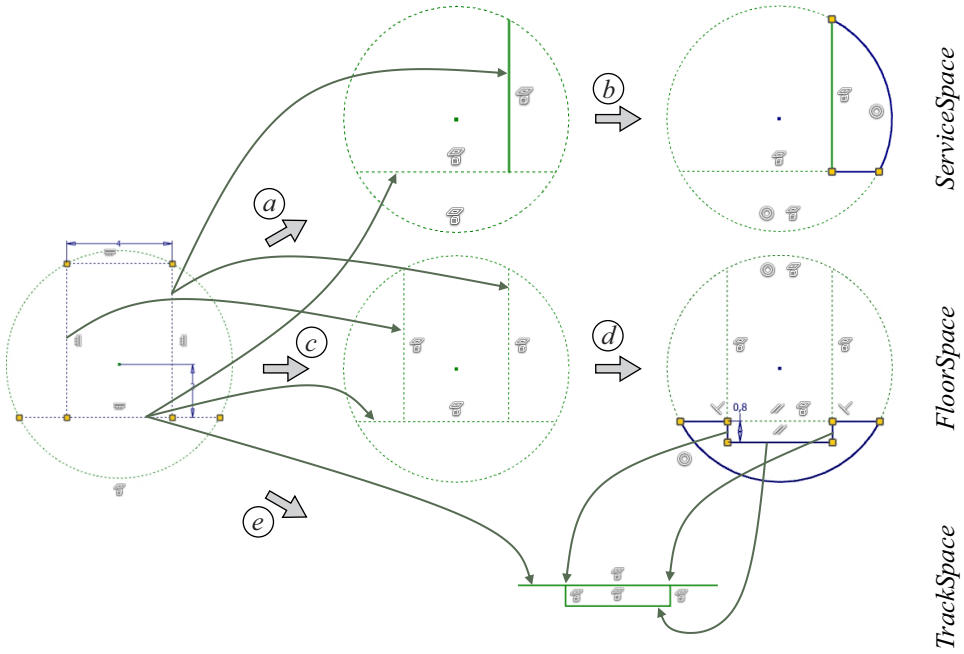


Abbildung 7.5: Visuelle Darstellung der Modellierungsoperationen zur Erstellung von *ServiceSpace*, *FloorSpace* und *TrackSpace* in Abhängigkeit der Konstruktionsgeometrie in der Skizze des *ClearanceSpace* (links). Die grünen Pfeile zeigen, welche geometrischen Objekte jeweils aus einer bereits erstellten Skizze projiziert wurden.

der Übersichtlichkeit dieses Abschnitts verzichtet. Eine vollständige Definition aller drei beschriebenen Regeln ist im Anhang B enthalten.

Nachdem der *ClearanceSpace* vollständig erstellt wurde, können die Skizzen zur Konstruktion der weiteren *Spaces* im LoD 4 erstellt werden. Da der *ClearanceSpace* als maßgebend betrachtet wird, werden die anderen *Spaces* in Abhängigkeit der Skizze dieses *Spaces* erstellt. Wird also die Breite des *ClearanceSpace* durch die Änderung des Parameterwerts der entsprechenden dimensionalen Zwangsbedingung verändert, passen sich die anderen *Spaces* entsprechend an. Die dafür notwendigen Modellierungsprozesse sind in Abbildung 7.5 abgebildet. Die Abhängigkeiten von *ServiceSpace*, *FloorSpace* und *TrackSpace* vom *ClearanceSpace* sind auch hier durch die Verwendung projizierter Geometrie gegeben. Dies ist in der Abbildung entsprechend hervorgehoben.

Für die Erstellung von *ServiceSpace* und *FloorSpace* sind jeweils zwei Graphersetzungsregeln definiert. Diese formalisieren die Modellierungsabläufe, die in Abbildung 7.5 mit *a* und *b* bzw. *c* und *d* gekennzeichnet sind. *a* und *b* bzw. *c* und *d* bilden jeweils als Paar den Modellierungsprozess zur Erstellung des *ServiceSpace* bzw. des *FloorSpace*, wobei in *a* und *c* zuerst die notwendige projizierte Geometrie erstellt wird und in *b* und *d* anschließend auf dieser Basis die notwendigen neuen geometrischen Elemente erzeugt werden. In Listing 7.6 ist die Graphersetzungsregel *LoD4_ServiceSpace_2* zur Repräsentation des in Abbildung 7.5 mit *b* gekennzeichneten Modellierungsablaufs dargestellt. Auf Basis der in Ablauf *a* bereits

erstellten projizierten Geometrie wird hier ein Kreisbogen und eine horizontale Linie erzeugt, die zusammen mit der projizierten vertikalen Linie, die nicht als Konstruktionsgeometrie angelegt wird, das Profil des *ServiceSpace* bilden. Die weiteren Graphersetzungsregeln sind in Anhang B enthalten.

```
rule LoD4_ServiceSpace_2{
    s:Sketch; if (s.name == "ServiceSpaceSketch");
    s-:contain->pc:ProjectedCircle; if (pc.name == "SS_circle");
    s-:contain->plb:ProjectedLine; if (plb.name == "SS_bottom");
    s-:contain->plr:ProjectedLine; if (plr.name == "SS_vline");
modify{
    s-:contain->arc:Arc; s-:contain->l:Line;
    plr-co1:coincident->l; plb-co2:coincident->l;
    plr-co3:coincident->arc; plb-co4:coincident->arc;
    pc-:concentric->arc;
eval{
    co1.port_s = PortType::StartPoint; co1.port_e = PortType::StartPoint;
    co2.port_s = PortType::EndPoint; co2.port_e = PortType::EndPoint;
    co3.port_s = PortType::EndPoint; co3.port_e = PortType::StartPoint;
    co4.port_s = PortType::EndPoint; co4.port_e = PortType::EndPoint;
    }
}}
```

Listing 7.6: Formale Definition der Graphersetzungsregel zur Durchführung des zweiten Modellierungsablaufs, der in Abbildung 7.5 mit *b* gekennzeichnet ist. Dies ist der zweite Ablauf im Modellierungsprozess zu Erstellung der Skizze, die die Basis für den *ServiceSpace* im LoD 4 bildet.

Nachdem alle Skizzen zur Erstellung der *Spaces* im LoD 4 erstellt wurden, müssen die entsprechenden Volumenkörper durch Sweeping Operationen erzeugt werden. Je *Space* ist dafür ein Modellierungsschritt notwendig. Hierzu wurde die Regel *SweepSketch* (Listing 7.7) so allgemein definiert, dass sie mehrmals ausgeführt werden kann, sodass auf Basis aller Skizzen eine Sweeping Operation durchgeführt wird. In der Regel ist weiterhin definiert, dass das *name*-Attribut des jeweiligen *Sweep*-Knotens auf Basis des *name*-Attributs des *Sketch*-Knotens, der die Basis der Sweeping Operation bildet, erstellt wird. Prinzipiell kann diese Regel daher durch geringfügige Anpassungen in verschiedenen Modellierungsszenarien eingesetzt werden.

In Abbildung 7.6 sind das finale mehrskalige Modell des Tunnels sowie der entsprechende Graph dargestellt. *FullTunnelSpace* und *InteriorSpace* sind in der Visualisierung des Modells hier nicht enthalten, da sie die anderen Teile des Modells komplett verdecken würden. Das erstellte parametrische Modell ist über verschiedene Parameterwerte veränderbar und überträgt Änderungen automatisch von niedrigeren in höhere LoDs. Es existiert natürlich noch eine Vielzahl weiterer Möglichkeiten zur Erstellung eines parametrischen Modells mit gleicher oder ähnlicher Funktionalität. Beispielsweise könnten in vielen Skizzen andere parametrische Zwangsbedingungen verwendet werden, die letztlich die gleiche Topologie

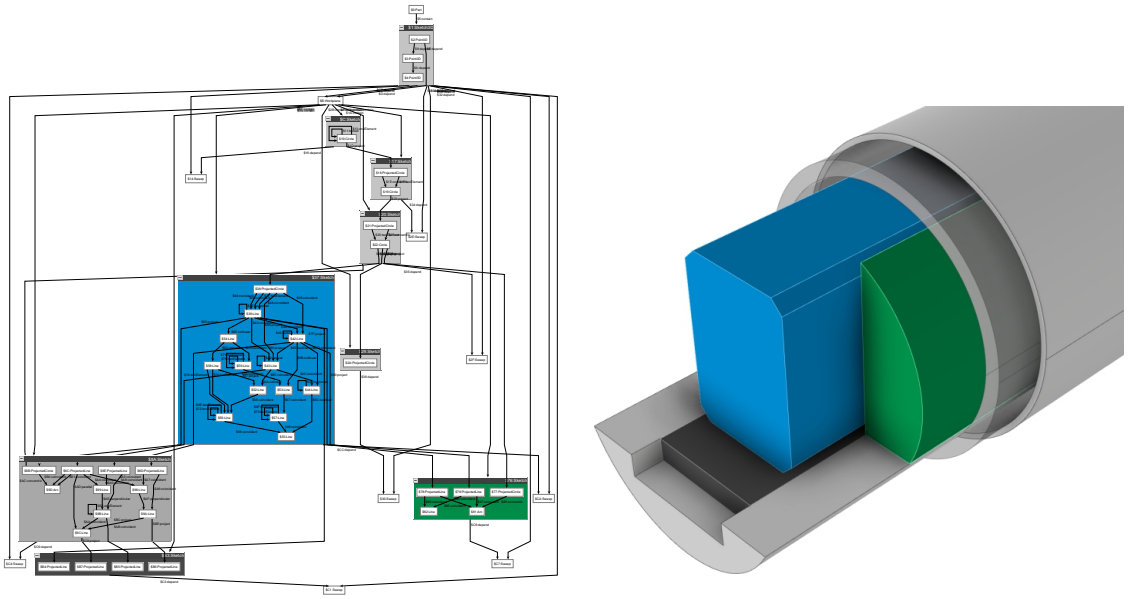


Abbildung 7.6: Durch die Ausführung der zuvor definierten Graphersetzungsregeln wird der hier dargestellte Graph mittels des Graphersetzungssystems erzeugt. Rechts ist das entsprechende parametrische Modell so visualisiert, dass die verschiedenen *Spaces* sichtbar sind. Zusätzlich wurden die Teile des Graphen, die die verschiedenen Skizzen im LoD 4 repräsentieren, entsprechend der Visualisierung des Modells eingefärbt.

definieren. Weiterhin ist es möglich, die hier definierten Graphersetzungsregeln so anzupassen, dass das parametrische Modell über andere Parameterwerte verändert werden kann oder beispielsweise in LoD 4 nicht der *ClearanceSpace* sondern der *TrackSpace* maßgebend für die Profile der anderen *Spaces* ist.

Das hier erstellte parametrische Modell und die dafür definierten Graphersetzungsregeln bilden die Grundlage für die Beispiele im nächsten Abschnitt, in dem die Modellierung eines Querschlags und eines Objekts im LoD 5 vorgestellt werden.

```
1 rule SweepSketch{
2     s3d:Sketch3D; if (s3d.name == "TunnelAxis"); s:Sketch;
3     negative {s-:depend->:SweepingOp;}
4 modify{
5     s3d-:depend->sw:Sweep<-:depend-s;
6 eval{
7     sw.name = s.name.substring(0,(s.name.length()-6));
8     }
9 }}
```

Listing 7.7: Graphersetzungsregel zur Durchführung der Sweeping Operationen zur Erstellung der Volumenkörper von *ClearanceSpace*, *ServiceSpace*, *FloorSpace* und *TrackSpace*. Diese Regel kann mehrmals angewendet werden, da mittels einer negativen Anwendungsbedingung verhindert wird, dass eine Skizze mehrfach als Basis eines Sweeps verwendet werden kann.

7.2 Tunnelquerschlag und Objekt in LoD 5

In diesem Abschnitt werden zwei Beispiele vorgestellt, die zusätzliche Bereiche des mehrskaligen Modells eines Schildvortriebstunnels zeigen, welche an bestimmten Positionen des Trassenverlaufs erstellt werden. Im Gegensatz zu den LoDs 1-4, die im vorigen Abschnitt betrachtet wurden, müssen zur Positionierung die konkreten Punkte, durch die die Trasse definiert ist, berücksichtigt werden. Dazu wird ein gerader Teilabschnitt eines Trassenverlaufs betrachtet, an den ein Querschlag angeschlossen werden soll. Zusätzlich wird an einer bestimmten Stelle des Trassenverlaufs ein Objekt erstellt, das dem LoD 5 zugeordnet ist.

In der folgenden Abbildung 7.7 sind die finalen Modelle dieser Beispiele dargestellt. Die Basis für beide Beispiele bildet der in der Abbildung links dargestellte Verlauf eines Tunnelabschnitts inkl. des Anschlusspunkts und des Verlaufs eines Querschlags. Der Verlauf des Tunnels ist dabei durch mehrere Punkte definiert, durch die die Stellen, an der Tunnel und Querschlag zusammentreffen oder an der ein Objekt im LoD 5 modelliert wird, festgelegt werden können.

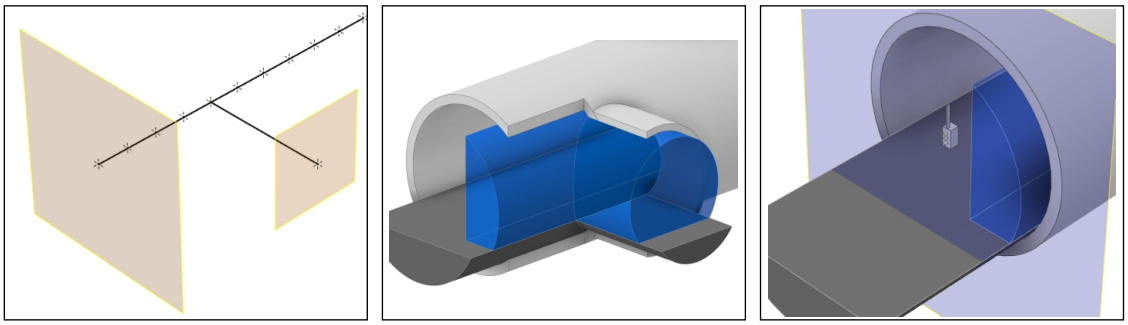


Abbildung 7.7: Trassenverlauf eines Tunnels und eines Querschlags (links) inkl. der davon abhängigen Arbeitsebenen. Modellierung des Anschlusses zwischen Tunnel und Querschlag (mitte, geschnittene Darstellung). Modellelement zur Repräsentation eines Signals im LoD 5 inkl. der dafür erstellten Arbeitsebene (rechts).

7.2.1 Querschlag

Ein Querschlag ist eine aus Sicherheitsgründen in regelmäßigen Abständen erforderliche Querverbindung zwischen zwei Tunnelröhren (Niklasch *et al.*, 2020). In diesem Beispiel wird gezeigt, wie der Anschluss eines Querschlags an den Haupttunnel in den LoDs 1-4 parametrisch modelliert werden kann. Die Detaillierungsgrade in denen der Querschlag dargestellt wird, orientieren sich hierbei an den für den Tunnel bereits definierten Detaillierungsgraden, wobei im LoD 4 nur der *ServiceSpace* und der *FloorSpace* betrachtet werden.

Die Modellierung der Verläufe von Tunnel und Querschlag entsprechen dabei im Wesentlichen den Abläufen, die bereits im vorangegangenen Abschnitt eingeführt wurden. Der Verlauf des

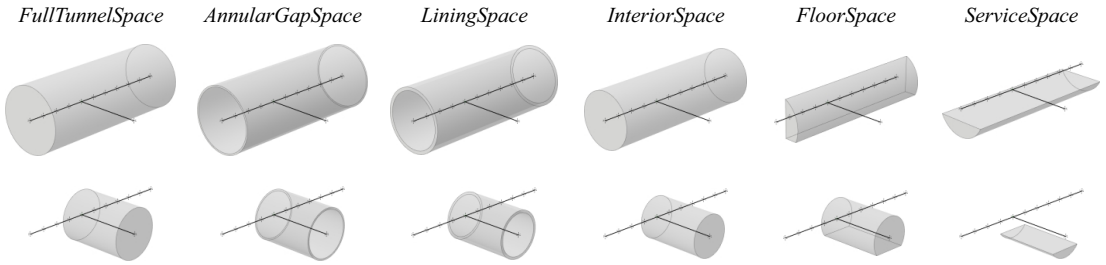


Abbildung 7.8: Alle betrachteten *Spaces* in den LoDs 1-4 in Tunnel (oben) und Querschlag (unten). Um diese Volumenkörper sinnvoll zusammenzufügen sind mehrere Boolesche Operationen notwendig.

Tunnelabschnitts ist in diesem Beispiel allerdings durch eine 3D-Skizze beschrieben, die den Trassenverlauf durch 3D-Punkte in regelmäßige Abstände segmentiert, wie in Abbildung 7.7 dargestellt. Wird das *name*-Attribut der *Point3D*-Knoten zur Repräsentation dieser 3D-Punkte mit bestimmten Werten belegt (z. B. mit einer Längenangabe zur Definition der Lage dieses Punkts auf der Trasse), können sich nachfolgende Modellierungsoperationen genau auf diesen Punkt beziehen.

An jeden dieser Punkte kann nun eine weitere 3D-Skizze angeschlossen werden, die den Verlauf des Querschlags kennzeichnet. Auf Basis dieser 3D-Skizzen können nun Arbeitsebenen erstellt werden, die als Grundlage für die Skizzen dienen, die wiederum die Grundlage für die Erstellung der Volumenkörper zur Beschreibung der *Spaces* von Tunnel und Querschlag sind. Die Modellierungsabläufe zur Erstellung der geometrischen Elemente innerhalb dieser Skizzen orientiert sich dabei stark an den bereits definierten Graphersetzungsregeln. Daher werden diese Regeln hier nicht im Detail wiederholt betrachtet, sondern vielmehr die darauf aufbauenden Modellierungsoperationen gezeigt, mit denen die Geometrie des Anschlusses von Tunnel und Querschlag erstellt wird. Werden Graphersetzungsregeln zur Erstellung der betrachteten *Spaces* auf die beiden 3D-Skizzen und die von ihnen abhängigen Arbeitsebenen angewendet, resultieren daraus die in Abbildung 7.8 dargestellten Volumenkörper. In Abbildung 7.9 ist der entsprechende Graph dargestellt.

Betrachtete man allerdings die Volumenkörper von Tunnel und Querschlag in Kombination ist direkt ersichtlich, dass hier mehrere Überschneidungen vorliegen. Dies ist in Abbil-

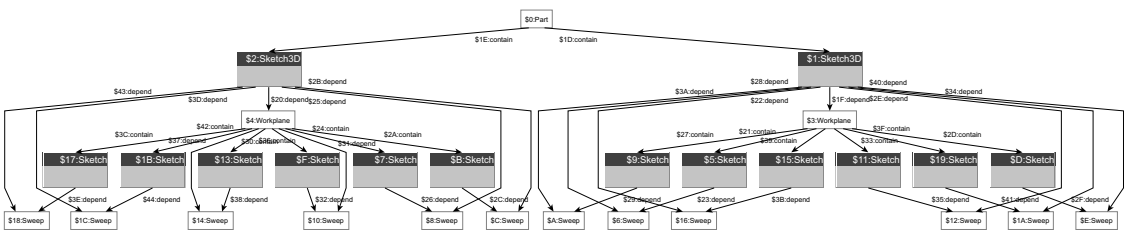


Abbildung 7.9: Graph, der alle Skizzen und Sweeping Operationen repräsentiert, die zur Erstellung der in Abbildung 7.8 dargestellten Volumenkörper beinhaltet. *Die geometrischen Elemente, die in den Skizzen enthalten sind und ihre Beziehungen zueinander sind hier aus Gründen der Übersichtlichkeit nicht dargestellt.*

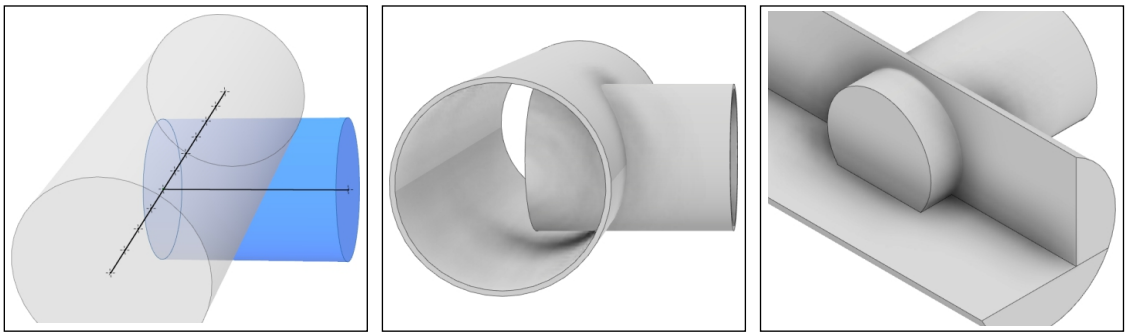


Abbildung 7.10: Darstellung der Verbindung von Querschlag und Tunnel im LoD 2 (links), LoD 3 (mitte) und im LoD 4 (rechts) während der Modellierung. Hier wurden noch keine Modellierungsoperationen durchgeführt, die die Geometrie des Anschlusses korrigieren.

dung 7.10 für verschiedene LoDs dargestellt. Diese Problematik bedingt sich dadurch, dass die Sweeping Operationen zur Erstellung der Volumenkörper des Querschlags den vollständigen Verlauf des Querschlags als Pfad referenzieren und somit bis in die Mitte des eigentlichen Tunnels hineinragen. Um diese Überschneidungen zu korrigieren sind mehrere Boolesche Operationnen notwendig, die einerseits die Teile des Querschlags, die in den Tunnel hineinragen, entfernen und weiterhin eine Öffnung im Tunnel erzeugen. Da so separate Volumenkörper für die *Spaces* von Tunnel und Querschlag erzeugt werden, ist auch im Nachhinein eine korrekte räumliche Aufteilung sichergestellt.

Abbildung 7.11 zeigt das entsprechende Vorgehen für den *FullTunnelSpace* im LoD 2. Hier wird der Volumenkörper des *FullTunnelSpace* des Tunnels vom Volumenkörper des *FullTunnelSpace* des Querschlags abgezogen. Das Resultat ist ein Volumenkörper, der die korrekte Geometrie des *FullTunnelSpace* des Querschlags beschreibt. Hierbei ist zu beachten, dass der *FullTunnelSpace* des Tunnels doppelt erstellt werden muss, da einer der beiden Volumenkörper nach Ausführung der Booleschen Operation nicht mehr vorhanden ist. So wird letztlich das in der Abbildung ganz rechts dargestellte Ergebnis, bestehend aus zwei separaten Volumenkörpern, erstellt.

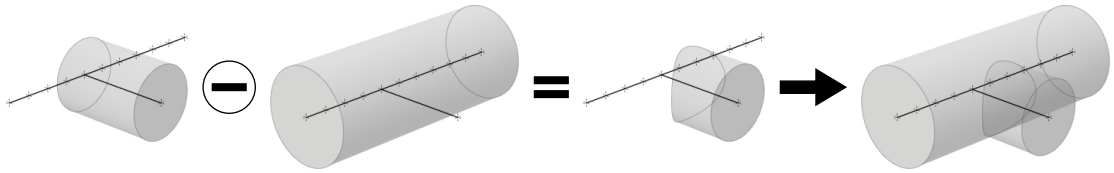


Abbildung 7.11: Boolesche Operation zur Erstellung des Volumenkörpers für den *FullTunnelSpace* des Querschlags. Hier muss der Volumenkörper des *FullTunnelSpace* des Tunnels von dem des Querschlags abgezogen werden. Rechts sind die beiden Volumenkörper, die den LoD 2 bilden, dargestellt.

```
rule LoD2_QS{
    FTS_Sketch:Sketch-:depend->FTS:Sweep<-:depend-TA:Sketch3D;
    if {FTS_Sketch.name == "FullTunnelSpaceSketch";}
    FTS_QS:Sweep;
    if {FTS_QS.name == "FullTunnelSpace_QS";}
modify{
    FTS_Sketch-:depend->FTS_temp:Sweep<-:depend-TA;
    diff:Difference; eval {diff.name = "FullTunnelSpace_QS";}
    FTS_QS-dep1:boolDepend->diff;
    FTS_temp-dep2:boolDepend->diff; eval {dep2.substract = true;}
    eval {FTS_QS.name = "";}
}}
```

Listing 7.8: Die hier definierte Graphersetzungsregel beschreibt den Modellierungsablauf, mit dem die korrekte Verschneidung der Volumenkörper der *FullTunnelSpaces* eines Tunnels und eines Querschlags in LoD 2 mittels Boolescher Operationen umgesetzt wird.

Die Graphersetzungsregel, mit der diese Boolesche Operation umgesetzt werden kann, ist in Listing 7.8 dargestellt. Wichtig ist hierbei, dass bei der Erstellung des Graphen, der die initialen Volumenkörper repräsentiert, auf eine korrekte Vergabe der *name*-Attribute der Skizzen, der Sweeping Operationen und der geometrischen Elemente geachtet wird. Dies muss bei der Definition entsprechender Graphersetzungsregeln berücksichtigt werden. Wurden die entsprechenden Knoten aber mit den korrekten *name*-Attributen versehen, so ist die Regel *LoD2_QS* flexibel anwendbar.

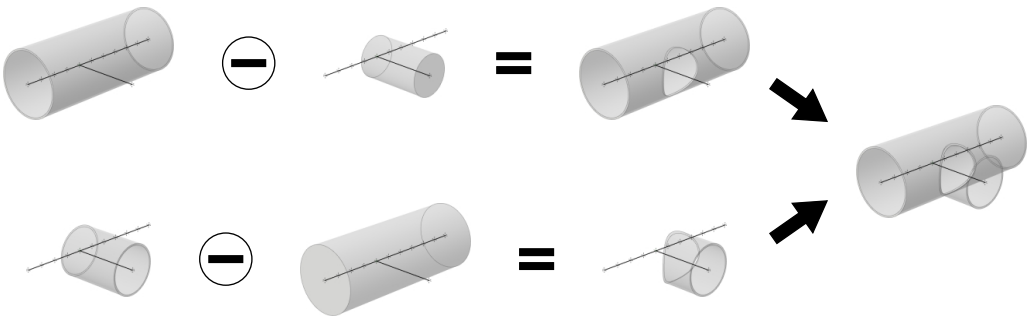


Abbildung 7.12: Boolesche Operation zur Erstellung der separaten Volumenkörper für den *AnnluarGapSpace* im Tunnel und Querschlag. Hier muss einerseits der *FullTunnelSpace* vom *AnnluarGapSpace* des Querschlags abgezogen werden (unten). Weiterhin muss ein zuvor erstellter Hilfskörper (der dem Hohlraum des *AnnularGapSpace* des Querschlags entspricht) vom *AnnluarGapSpace* des Tunnels abgezogen werden, um die Öffnung zum Anschluss des Querschlags zu erzeugen (oben). Links sind die beiden Volumenkörper, die den *AnnluarGapSpace* im LoD 3 bilden, dargestellt.

In Abbildung 7.12 ist das entsprechende Vorgehen für den *AnnluarGapSpace* im LoD 3 dargestellt. Hier muss einerseits ein zuvor erstellter Hilfskörper (der dem Hohlraum des *AnnularGapSpace* des Querschlags entspricht) vom *AnnluarGapSpace* des Tunnels abgezogen werden, um die Öffnung zum Anschluss des Querschlags zu erzeugen. Konkret bedeutet dies, dass der innere Kreis in der Skizze des *AnnluarGapSpace* in eine neue Skizze

projiziert werden muss. Ein Sweep auf Basis dieser Skizze erzeugt dann den Hilfskörper, der vom *AnnluarGapSpace* des Tunnels abgezogen wird. Weiterhin muss der *FullTunnelSpace* des Tunnels vom *AnnluarGapSpace* des Querschlags abgezogen werden (unten). Hierzu wird ein weiterer Hilfskörper, der dem *FullTunnelSpace* des Tunnels aus LoD 2 entspricht erstellt. Links in Abbildung 7.12 sind die beiden Volumenkörper dargestellt, die den *AnnluarGapSpace* im LoD 3 bilden.

Die Booleschen Operationen zur Erzeugung der Volumenkörper von *LiningSpace*, *InteriorSpace*, *ServiceSpace* und *FloorSpace* erfolgen grundsätzlich nach dem gleichen Prinzip und sind hier nicht mehr im Detail abgebildet. In Abbildung 7.13 sind die finalen Volumenkörper von *AnnluarGapSpace*, *LiningSpace* und *InteriorSpace* dargestellt. Auf der linken Seite sind die Volumenkörper geschnitten, sodass die korrekte Ausgestaltung der Geometrie sichtbar wird. In der zu Beginn dieses Abschnitts aufgeführten Abbildung 7.10 sind auch die *Spaces* im LoD 4 visualisiert.

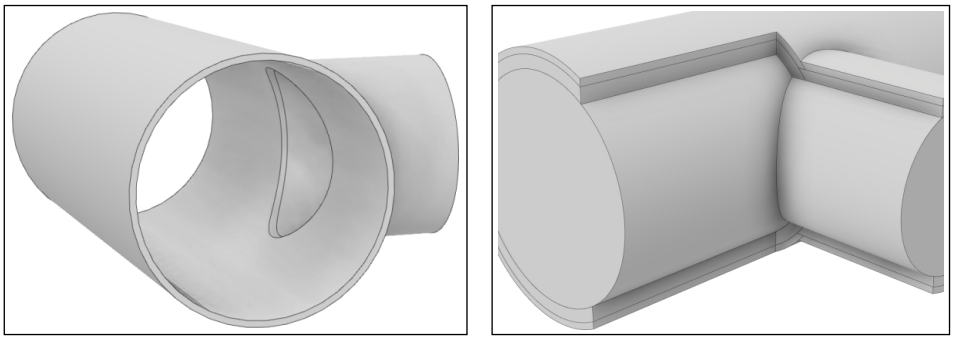


Abbildung 7.13: Finale Volumenkörper des *LiningSpace* (links) und aller *Spaces* in den LoDs 2 und 3 (rechts). In Abbildung 7.10 sind auch die *Spaces* im LoD 4 visualisiert.

Die Graphersetzungsregeln, die die Verschneidungsoperationen in den LoDs 2-4 repräsentieren, sind in Anhang B vollständig aufgeführt. Die Regeln sind dabei nach den einzelnen LoDs aufgeteilt, sodass mit der Anwendung einer Regel die Volumenkörper von Tunnel und Querschlag in einem LoD vollständig bearbeitet werden. Im Vergleich zur manuellen Modellierung ist die Durchführung mittels dieser Regeln deutlich weniger aufwändig, da bei der manuellen Modellierung die einzelnen Volumenkörper mehrfach ein- und ausgeblendet werden müssen. Dies zeigt den Vorteil, der durch die Formalisierung der entsprechenden Modellierungsoperationen erzielt werden kann. Es muss allerdings angemerkt werden, dass für die Anwendung der verschiedenen Regeln eine korrekte Benennung der verschiedenen Elemente zwingend erforderlich ist, und die Regeldefinitionen bei der Verwendung anderer Namen für die verschiedenen *Spaces* entsprechend angepasst werden müssen.

In Abbildung 7.14 ist abschließend der finale Graph, der das Modell von Tunnel und Querschlag repräsentiert, dargestellt. Die Darstellung wurde dabei so angepasst, dass nur die geometrischen Elemente der Skizzen enthalten sind, die für dieses Beispiel relevant sind.

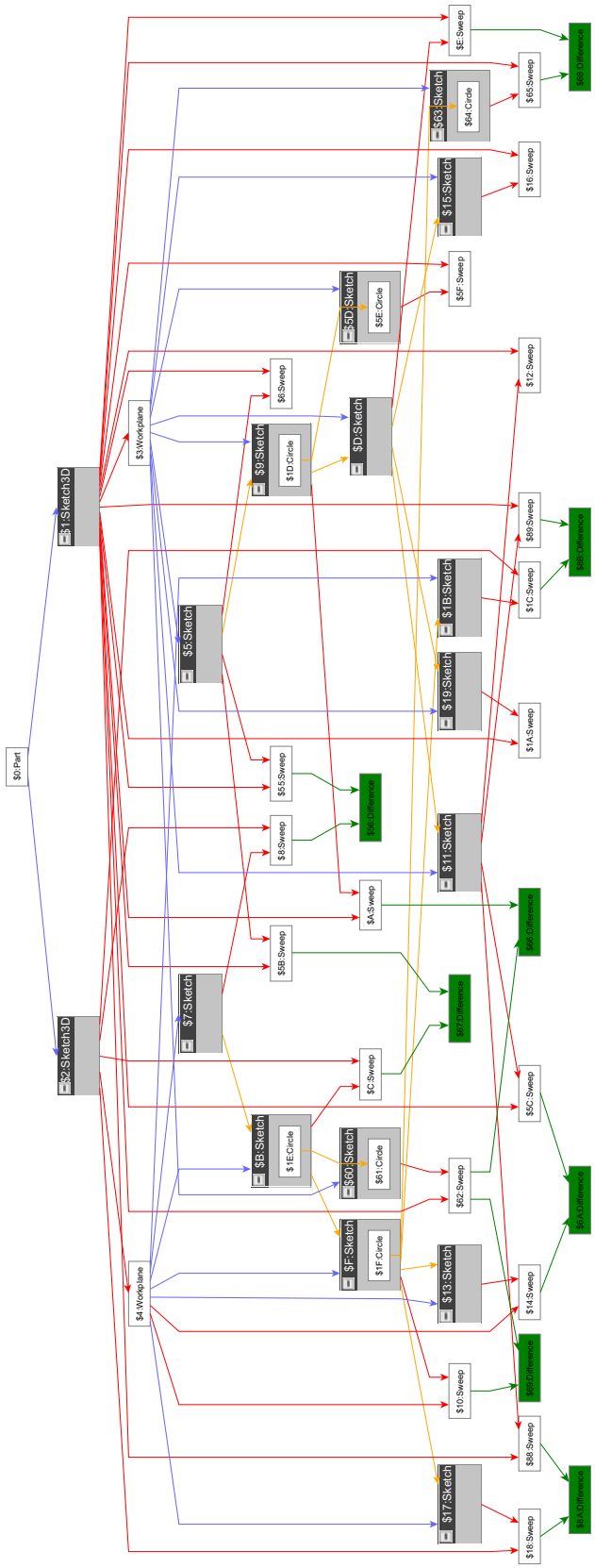


Abbildung 7.14: Zustand des Graphen nach Anwendung der Ersetzungsregeln zur korrekten Verschneidung der verschiedenen Volumenkörper. Die Kanten sind entsprechend ihres Typs eingefärbt: *contain* (blau), *depend* (rot), *boolDepend* (grün), *project* (orange). Weiterhin sind die *Difference*-Knoten grün eingefärbt. *In dieser Darstellung des Graphen sind nur die geometrischen Elemente der Skizzen enthalten, die für die Anwendung der hier betrachteten Regeln relevant sind.*

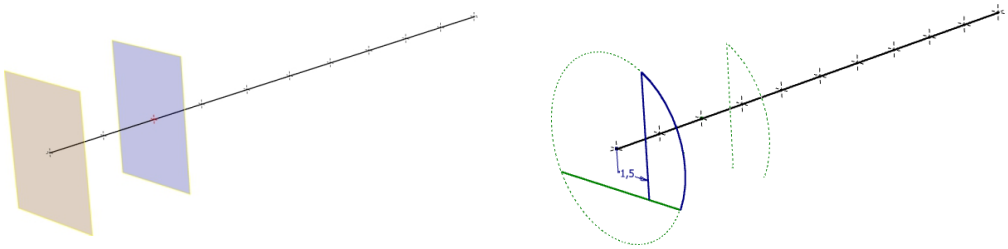


Abbildung 7.15: Links: Erstellung der Arbeitsebene in Abhängigkeit eines Punkts auf der Trasse. Rechts: Projektion der vertikalen Linie und des Kreisbogens aus dem *ServiceSpace* in die auf dieser Arbeitsebene erstellte Skizze.

7.2.2 Objekt im LoD 5

Entsprechend der in diesem Kapitel bereits beschriebenen fünf Detaillierungsgrade eines Schildvortriebstunnels, die durch Borrmann *et al.* definiert wurden, werden physikalische Objekte erst im fünften LoD modelliert. In den niedrigeren LoDs 1-4 wird lediglich die räumliche Aufteilung des Bauwerks durch die verschiedenen *Spaces* beschrieben. Im folgenden Beispiel wird ein Objekt, das innerhalb des in LoD 4 definierten *ServiceSpace* verortet ist, erstellt und so positioniert, dass es sowohl vom Verlauf der Tunneltrasse als auch von der Geometrie des *ServiceSpace* parametrisch abhängig ist. Das Vorgehen bei der Modellierung gliedert sich in drei wesentliche Schritte:

- Erstellung einer Arbeitsebene an der Position des Bauwerks, an der das Objekt erstellt werden soll. Die Arbeitsebene ist dabei von einem der Punkte, durch den sich der Trassenverlauf definiert und zusätzlich vom Trassenverlauf selbst, abhängig. So wird die Arbeitsebene an der richtigen Position orthogonal zum Trassenverlauf erstellt. Auf dieser Arbeitsebene wird im Anschluss eine Skizze erstellt.

- In diese Skizze werden anschließend Teile der Geometrie des *ServiceSpace* projiziert. Damit kann die Position des Objekts von den Abmessungen des *ServiveSpace* parametrisch abhängig gemacht werden, sodass sich Änderungen an den *Spaces* im LoD 4 automatisch auf das Objekt übertragen.

- Auf Basis der projizierten Geometrie kann nun das Profil des Objekts skizziert werden. Diese Skizze bildet die Basis für die Erzeugung des eigentlichen Objekts als Volumenkörper durch eine Extrusion. Konkret wird hier ein Modellelement zur Repräsentation eines Signals betrachtet.

In Abbildung 7.15 sind die ersten beiden genannten Schritte visuell dargestellt. Die Graphersetzungsregel, die den Modellierungsablauf zur Erstellung der Arbeitsebene und der Skizze repräsentiert, ist in Listing 7.9 aufgeführt. Der 3D-Punkt der zur Positionierung genutzt werden soll, ist in diesem Beispiel durch eine zusätzlich im Graphen hinterlegte

Eigenschaft (siehe auch Abschnitt 6.4.2, Seite 170), die mittels eines *TextualAtt*-Knotens definiert wurde, gekennzeichnet. Das *name*- und das *value*-Attribut dieses Knoten müssen mit den Werten *has_object* bzw. *signal* versehen sein, damit der in der Ersetzungsregel definierte Mustergraph eine Passung im Arbeitsgraphen identifiziert. Diese Regel kann somit auch mehrfach angewendet werden, um an allen entsprechend gekennzeichneten Punkten auf der Trasse eine Arbeitsebene und eine Skizze zu erstellen.

```
rule LoD5_WP{
    s3d:Sketch3D;
    p:Point3D-:attribute->tx:TextualAtt;
    if {tx.name == "has_object"; tx.value == "signal";}
    s3d-:contain->p;
modify{
    wp:Workplane;
    s3d-:depend->wp;
    p-:depend->wp;
    wp-:contain->s:Sketch;
eval{
    wp.type = WpType::context;
    s.name = "SignalSketch";
    }
}}
```

Listing 7.9: Graphersetzungsregel, die an einem Punkt auf einer Tunneltrasse eine Arbeitsebene erstellt und zusätzlich eine leere Skizze auf dieser Arbeitsebene anlegt.

```
rule LoD5_project{
    s:Sketch;
    if {s.name == "SignalSketch";}
    ss:Sketch;
    if (ss.name == "ServiceSpaceSketch");
    ss-:contain->a:Arc;
    ss-:contain->:ProjectedLine<-:project-l_cs:Line;
    if {l_cs.name == "L_right";}
modify{
    s-:contain->pa:ProjectedArc<-:project-a;
    s-:contain->pl:ProjectedLine<-:project-l_cs;
eval{
    pa.construction = true;
    pl.construction = true;
    }
}}
```

Listing 7.10: Graphersetzungsregel, mittels der bestehende geometrische Elemente als Referenz für das neu zu erstellende Objekt in die Skizze *SignalSketch* projiziert werden.

Listing 7.10 zeigt die Graphersetzungsregel, die in der so erstellten Skizze die geometrischen Elemente des *ServiceSpace* (in Abbildung 7.15 rechts dargestellt), projiziert. Konkret werden

ein *ProjectedLine*- und ein *ProjectedArc*-Knoten erstellt. Die Geometrie, die projiziert werden soll, wird mittels *project*-Kanten referenziert. Beide geometrischen Elemente werden als Konstruktionsgeometrie erzeugt, da sie bei einer nachfolgenden Extrusion der Skizze nicht berücksichtigt werden sollen. Tatsächlich wäre für die Positionierung des Signals bereits die Projektion der Linie ausreichend – sollen aber noch weitere Objekte erstellt werden, ist es potenziell hilfreich, auch den Kreisbogen zu referenzieren. Letztlich wird die Regel so auch für alternative Modellierungsprozesse nutzbar.

Durch Nutzung der projizierten Geometrie als Referenz können nun neue geometrische Elemente in der Skizze erstellt und mittels parametrischer Zwangsbedingungen korrekt positioniert werden. In einem ersten Modellierungsablauf wird dazu ein parametrisiertes Rechteck erstellt und auch direkt extrudiert. Breite und Höhe dieses Rechtecks werden mittels dimensionaler Zwangsbedingungen festgelegt. Die Graphersetzungsregel *LoD5_rectangle_ex* ist in Listing B.9 in Anhang B aufgeführt. Die Regel entspricht dabei im Wesentlichen der bereits im vorangegangenen Kapitel beschriebenen Regel zur Erstellung eines Quadrats (siehe Listing 6.7, Abschnitt 6.5.2).

```
1  rule LoD5_position{
2      s:Sketch;
3      if {s.name == "SignalSketch";}
4      s-:contain->pl:ProjectedLine<-:project-;
5      s-:contain->l2:Line; if {l2.name == "left";}
6  modify{
7      pl-:collinear->l2;
8      pl-dim:twoElement->l2;
9  eval{
10     dim.value = 0.8;
11     dim.mode = ModeType::Direct;
12     dim.port_s=PortType::EndPoint; dim.port_e=PortType::EndPoint;
13     }
14 }}
```

Listing 7.11: Graphersetzungsregel, mittels der ein zuvor modelliertes Rechteck an der projizierten Konstruktionsgeometrie ausgerichtet wird.

Zu beachten ist, dass die projizierte Geometrie für die initiale Erstellung des Rechtecks hier noch nicht berücksichtigt wird. Dies erfolgt erst im nachfolgenden Modellierungsablauf (Regel *LoD5_position*, Listing 7.11), in dem das Rechteck mittels einer dimensionalen und einer *kollinear*-Zwangsbedingung an der projizierten vertikalen Linie ausgerichtet wird. Dies ist in Abbildung 7.16 (links) dargestellt. Die Skizze ist damit vollständig parametrisiert. Grundsätzlich könnte hier auch eine alternative Regel zur Positionierung erstellt werden, die das Signal beispielsweise mit einem bestimmten Abstand zur Begrenzung des *ServiceSpace* positioniert.

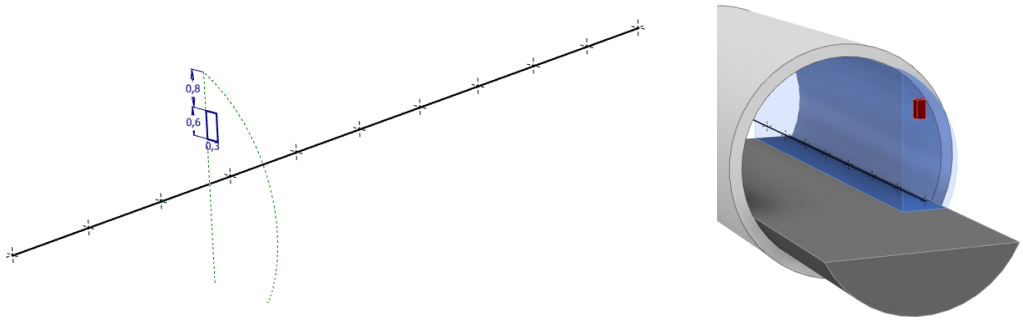


Abbildung 7.16: Links: Position des Rechtecks nach Ausrichtung an der projizierten Geometrie. Rechts: Darstellung des Signals im Kontext des gesamten Modells.

In Abbildung 7.17 ist abschließend der finale Graph, der das erstellte Objekt repräsentiert, abgebildet. Hier ist auch direkt erkennbar, dass der *ProjectedCircle*-Knoten eigentlich nicht benötigt wird, da er keine Verbindung zu den anderen geometrischen Elementen aufweist. Die hier beschriebenen Regeln lassen sich zu einem Modellierungsprozess zusammenfassen und können wiederholt angewendet werden. So ist es möglich, nach einmaliger Definition der Graphersetzungsregeln Objekte im LoD 5 an verschiedenen Positionen im Modell des Tunnels zu erstellen.

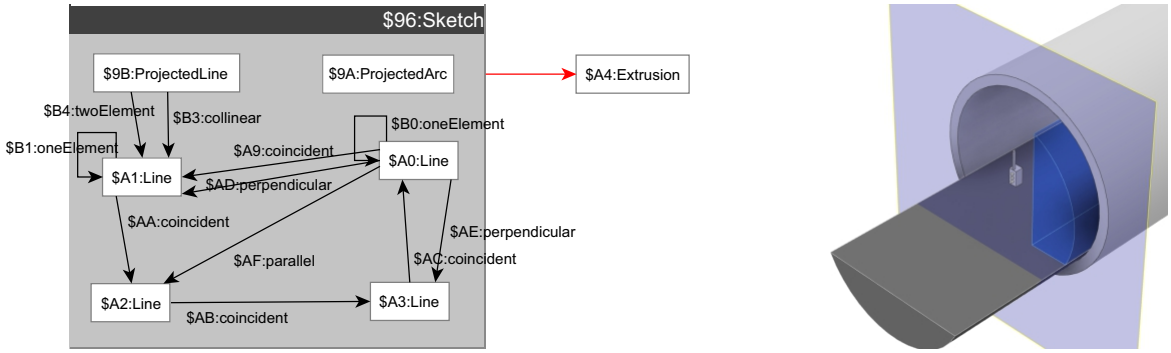


Abbildung 7.17: Rechts: Teil des finalen Graphen, der die Geometrie des Signals repräsentiert. Links: Mögliche weitere Detaillierung des Volumenkörpers zur Repräsentation des Signals.

7.3 Modellierung von Profilträgern

Im letzten Beispiel dieses Kapitels wird gezeigt, wie die parametrische Modellierung der Profile von Stahlträgern durch Graphersetzungsregeln formalisiert werden kann. Hierzu wird das in Abschnitt 2.2.2 beschriebene Konzept der Levels of Development (LOD) aufgegriffen und das Modell zweier Profilträger vom LOD 100 in den LOD 200 detailliert. Anschließend wird dargestellt, wie zwei Träger, die als separate Bauteile modelliert wurden, in einer Baugruppe durch Mating-Zwangsbedingungen aneinander ausgerichtet werden können.

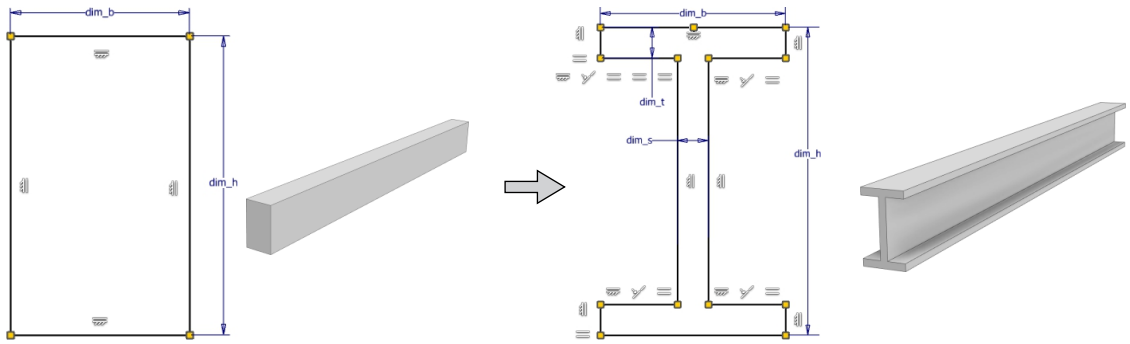


Abbildung 7.18: Detaillierung einer Skizze als Basis einer Extrusion zur Erstellung eines Volumenkörpers, der einen I-Träger darstellt.

Abbildung 7.18 zeigt die parametrisierten Skizzen eines Stahlprofils im LOD 100 (links) und LOD 200 (rechts). Die Graphersetzungsregel zur Erstellung des rechteckigen Querschnitts im LoD 100, der mittels der Parameter *dim_b* und *dim_h* dimensioniert werden kann, ist in Anhang B, Listing B.10 enthalten. Die Regel beinhaltet auch die Extrusion des Trägers. Das so erstellte Profil kann mit einer weiteren Graphersetzungsregel detailliert werden, sodass das Profil des Trägers im LOD 2 modelliert wird. Neben den Parametern, mittels derer Breite und Höhe des Trägers gesteuert werden können, sind nun auch Parameter für die Dicke des Flansches und die Stärke des Stegs Teil des parametrischen Modells. Die verwendeten Parameter zur Beschreibung des Profils orientieren sich an den Abmessungen, mit denen IPE-Träger im Allgemeinen beschrieben werden. Durch die Änderung dieser Parameterwerte kann das Profil so angepasst werden, dass es einem beliebigen IPE-Profil entspricht. Alternativ können die Werte der Parameter auch durch eine Graphersetzungsregel angepasst werden, die die Werte der Parameter entsprechend des gewünschten Profiltyps festlegt.

Die Graphersetzungsregel zur Detaillierung der Skizze des Profils wurde in diesem Beispiel so konzipiert, dass die Abmessungen des Trägers im LOD 200 so gewählt werden, dass sie den Abmessungen der Repräsentation des Trägers im LOD 100 entsprechen. Technisch gesehen werden dazu die temporären Koordinaten der Linien in der Skizze des LOD 200 auf Basis der Abmessungen des Rechtecks in LOD 100 berechnet. Neben den Linien, die das Profil des Trägers beschreiben, wird in LOD 200 ein zusätzlicher Punkt in die Skizze eingefügt, der koinzident zum Mittelpunkt der oberen Begrenzungslinie des oberen Flansches ist. Dieser Punkt ist für die folgende Ausrichtung von zwei mittels dieser Regeln erstellten Träger-Bauteile notwendig.

Abbildung 7.19 zeigt, wie diese Ausrichtung der beiden Bauteile in einer Baugruppe umgesetzt wird. Mittels dreier Mating-Zwangsbedingungen werden die gekennzeichneten Mittelpunkte koinzident gesetzt und die oberen Begrenzungslinien des Trägerprofils als kollinear definiert. Zusätzlich wird die Fläche, die sich durch die Extrusion der oberen

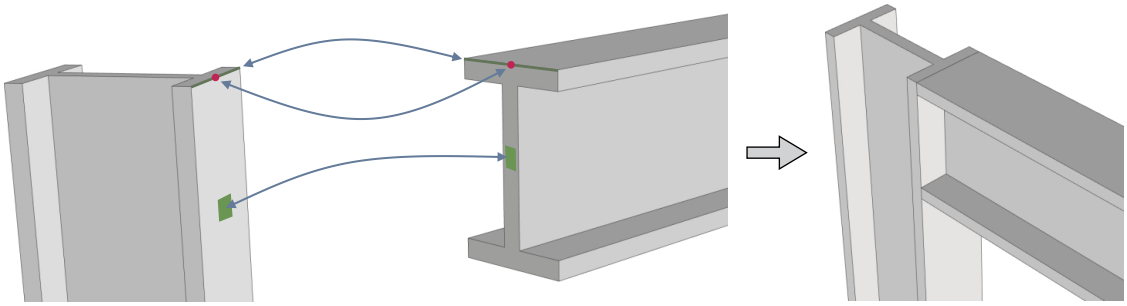


Abbildung 7.19: Ausrichtung zweier Stahlprofile mittels verschiedener Mating-Zwangsbedingung.

Begrenzungslinie des einen Trägers ergibt, mit der Grundfläche des zweiten Trägers als komplanar definiert.

Damit sind die beiden Träger fest aneinander ausgerichtet. Die entsprechende Graphersetzungsregel ist in Listing 7.12 definiert. Hierbei ist zu beachten, dass im Mustergraph der Graphersetzungsregel nicht mittels eines *name*-Attributs konkret festgelegt werden muss, welche *Sketch*-Knoten in Zeile 2 und 3 als Passung im Arbeitsgraphen identifiziert werden sollen. Dies bedingt sich durch das in GrGen.NET standarmäßig verwendete *isomorphic matching*. Das bedeutet, dass für *s1* und *s2* nicht dieselben *Sketch*-Knoten im Arbeitsgraphen als Passung gefunden werden dürfen.

```
1  rule mate_beams{
2      s1:Sketch-:contain->p1:Point;
3      s2:Sketch-:contain->p2:Point;
4      s1-:contain->l1:Line; if (l1.name == "Flansch_L0");
5      s2-:contain->l2:Line; if (l2.name == "Flansch_L0");
6  modify{
7      s1-:depend->plane1:DerivedGeo;
8      s2-:depend->plane2:DerivedGeo;
9      plane1-:mCoplanar->plane2;
10     p1-:mCoincident->p2;
11     l1-col:mCollinear->l2;
12 }}
```

Listing 7.12: Graphersetzungsregel, mittels der zwei Stahlprofile aneinander ausgerichtet werden.

Der mit Hilfe der Graphersetzungsregeln erzeugte Graph ist in Abbildung 7.20 dargestellt. Zusätzlich ist das entsprechende Modell abgebildet. Im Graphen sind die Kanten, mittels derer die Mating-Zwangsbedingungen repräsentiert werden, direkt als Querverbindungen zwischen den als Gruppe dargestellten *Sketch*-Knoten erkennbar.

An diesem Beispiel lässt sich bereits einer der Punkte, die im nächsten Kapitel im Rahmen der Diskussion der Ergebnisse aufgegriffen werden, herausarbeiten. Dies betrifft die Ausrichtung der verschiedenen Bauteile mittels der Mating-Zwangsbedingungen. Bei einer manuellen Modellierung in einer CAD-Anwendung ist dies durch die visuelle Darstellung des Modells

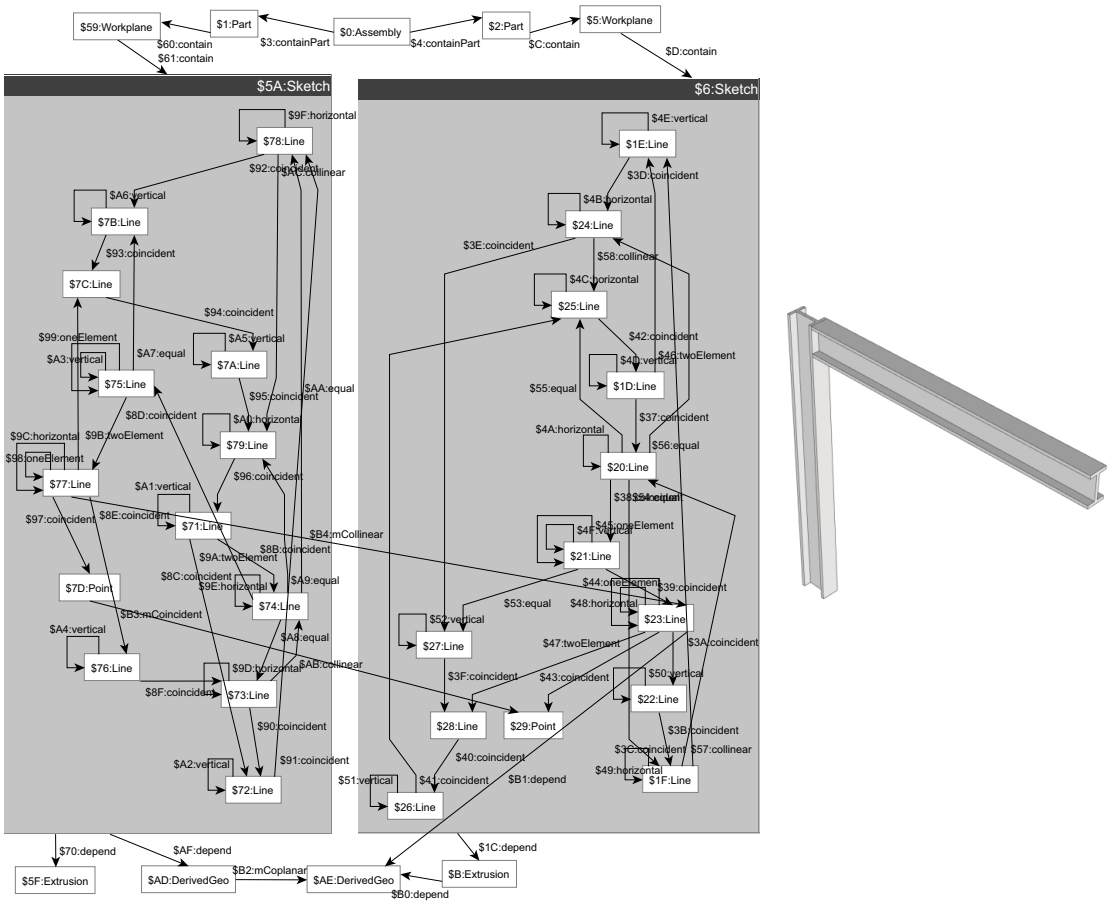


Abbildung 7.20: Graphbasierte Repräsentation der beiden Träger, die mittels Mating-Zwangsbedingungen aneinander ausgerichtet wurden (links). Rechts ist das so erstellte Modell abgebildet.

weitgehend intuitiv umsetzbar. Die Punkte, Linien und Flächen, die ausgerichtet werden sollen, können durch den Nutzer direkt ausgewählt werden, ohne dass berücksichtigt werden muss, wie die entsprechenden geometrischen Elemente ursprünglich erstellt wurden. Auch das Resultat der Ausrichtungen ist direkt sichtbar. Gerade bei komplexeren Geometrien kann die manuelle Ausführung aber durchaus mit einem gewissen Aufwand verbunden sein, da jeweils die korrekten Elemente ausgewählt werden müssen, was eine entsprechende Ausrichtung der 3D-Visualisierung des Modells erfordert. Wird diese Ausrichtung der Elemente aneinander hingegen mittels formalisierter Regeln durchgeführt, ist es nicht möglich, intuitiv die richtigen geometrischen Elemente auszuwählen. Vielmehr müssen alle Elemente, die potenziell ausgerichtet werden sollen, so benannt werden, dass sie zu einem späteren Zeitpunkt eindeutig identifizierbar sind. Dies stellt einen initialen Aufwand dar, der bei einer manuellen Modellierung so nicht notwendig ist. Demgegenüber steht die mögliche Automatisierung der Ausrichtungsoperationen und die damit verbundene Reduktion des Aufwands im Vergleich zu einer manuellen Ausrichtung.

Dies zeigt letztlich, dass bei einer Automatisierung immer der initiale Aufwand im Kontext der späteren Arbeitserleichterung betrachtet werden muss.

7.4 Zusammenfassung

In diesem Kapitel wurde anhand von Fallstudien herausgearbeitet, wie Modellierungsschritte in einem praktischen Kontext durch Graphersetzungsregeln ausgedrückt werden können. Dazu wurden drei Szenarien, in denen ein Graphersetzungssystem zur parametrischen Modellierung eingesetzt werden kann, eingeführt und entsprechende Graphersetzungsregeln zur Erweiterungen des Graphersetzungssystems definiert.

Dadurch wurde gezeigt, dass Modellierungsoperationen in verschiedenen Kontexten durch Graphersetzungsregeln formalisiert werden können und die Möglichkeit zur Erweiterung des Graphersetzungssystems prinzipiell gegeben ist. Der vorgestellte Ansatz ist damit grundsätzlich in verschiedenen Bereichen anwendbar und kann zur Formalisierung von Ingenieurwissen im Kontext der parametrischen Modellierung genutzt werden.

Kapitel 8

Zusammenfassung, Diskussion und Ausblick

Die vorliegende Arbeit hat die Unterstützung und Teilautomatisierung der Modellierungsprozesse von dreidimensionalen parametrischen Bauwerksmodellen intensiv untersucht. Auf dieser Basis wurde ein Ansatz vorgestellt, der es ermöglicht, die Abfolge von konsekutiven oder konkurrierenden Modellierungsoperationen formalisiert zu definieren und bei Bedarf abzurufen und automatisiert auszuführen. So können Modellierungsprozesse formalisiert und in verschiedenen Modellierungsszenarien angewendet werden. Weiterhin ist es möglich mit diesem Ansatz auch die Qualität der erstellten digitalen Modelle zu verbessern, da durch die Anwendung der formalisierten Modellierungsschritte Fehler vermeidbar werden, die bei einer rein manuellen Modellierung auftreten können.

Zur formalen Definition und Speicherung der Modellierungsoperationen wurden Graphersetzungsregeln eingesetzt. Diese Graphersetzungsregeln wurden so konzipiert, dass nicht nur eine Änderung und Erweiterung der geometrischen Elemente erfolgt, sondern auch die parametrischen Zusammenhänge des Modells entsprechend der Regeldefinition verändert und erweitert werden. Auch nach einer graphbasierten Modellerstellung können die Modelle dementsprechend über die Veränderung der Parameterwerte an veränderte Randbedingungen angepasst werden. Ein grundlegender Beitrag dieser Arbeit besteht in der Definition einer graphbasierten Repräsentation parametrischer Modelle, da nur ein in Form eines Graphen gespeichertes parametrisches Modell auch durch eine Graphersetzungsregel verändert werden kann. Dies ist die Grundlage zur Definition und Anwendung von

Graphersetzungsregeln zur formalen Beschreibung der Modellierungsoperationen, die den wesentlichen Beitrag dieser Arbeit ausmacht.

Die praktische Anwendungsmöglichkeit der vorgestellten Methode wurde in verschiedenen Szenarien im Kontext der mehrskaligen Modellierung und der Detaillierung von Modellen erläutert. Die vorgestellten Fallstudien zeigen, wie Modellierungsprozesse, die sonst manuell in einer spezifischen Software durchgeführt werden, durch Graphersetzungsregeln herstellerneutral formalisiert werden können.

Damit wurden die in Abschnitt 1.2 formulierten Fragestellungen im Rahmen dieser Arbeit intensiv und detailliert untersucht. Es wurde gezeigt, wie parametrische Modelle entsprechend der Anforderungen in Kapitel 5 durch einen Graph repräsentiert und durch ein Graphersetzungssystem formal verändert werden können. Die Repräsentation dieser parametrischen Modelle durch einen Graphen und die Formulierung von Graphersetzungsregeln, aus denen sich das Graphersetzungssystem zusammensetzt wurden in Kapitel 6 ausführlich beschrieben. Weiterhin wurde in diesem und dem folgenden Kapitel 7 gezeigt, dass die durch den Graphen repräsentierten Modelle den Anforderungen aus Kapitel 5 entsprechen, durch Graphersetzungsregeln formal veränderbar sind und somit interpretiert werden können um parametrische Modelle zu erzeugen.

Im Folgenden wird die letzte der in Abschnitt 1.2 formulierten Fragestellungen aufgegriffen und die Ergebnisse dieser Arbeit werden hinsichtlich ihrer Einsatzmöglichkeiten und Grenzen diskutiert. Anschließend wird ein Ausblick zu Möglichkeiten der Weiterentwicklung des Ansatzes und hinsichtlich des allgemeinen Entwicklungsbedarfs im Kontext dieser Arbeit gegeben.

8.1 Diskussion der Einsatzmöglichkeiten und Grenzen

Grundsätzlich zeigen die Beispiele des vorangegangenen Kapitels, dass es möglich ist, die einzelnen Modellierungsoperationen eines parametrischen Modellierungsprozesses formal durch Graphersetzungsregeln zu definieren. Durch ein Ausführen dieser Graphersetzungsregeln und eine anschließende Interpretation des erzeugten Graphen können verschiedene parametrische Modelle generiert werden. In diesem Abschnitt wird der entwickelte Ansatz diskutiert und kritisch betrachtet. Dabei wird einerseits auf die potenziellen Möglichkeiten, die sich durch die entwickelte Methode ergeben, eingegangen. Andererseits werden die Grenzen des Ansatzes beschrieben sowie Aufwand und Nutzen gegenübergestellt.

Bereits in Lee *et al.* (2006b) wurde das Potenzial der parametrischen Modellierung zur Einbettung von Ingenieurwissen in Bauwerks- und Gebäudemodelle erkannt und eine Methode beschrieben, mit der die Zusammenhänge eines parametrischen Modells mittels einer graphischen Notation beschrieben werden können. Ansätze zur vollständigen Formalisierung

der parametrischen Zusammenhänge oder eine automatisierte Erstellung und Veränderung der so beschriebenen Modelle wurden allerdings nicht betrachtet. Ein wesentlicher Beitrag dieser Arbeit besteht daher in der Entwicklung einer Methode, die es ermöglicht, Wissen zu parametrischen Modellierungs- und Detaillierungsvorschriften und -prozessen formal durch Graphersetzungsregeln zu beschreiben. Ein entsprechend geschulter Nutzer hat damit die Möglichkeit, sein Wissen zu parametrischen Modellierungsoperationen so abzubilden, dass sie im Anschluss wiederholt ausgeführt werden können, um einen Graphen zu erzeugen, der das durch diese Modellierungsoperationen erzeugte parametrische Modell repräsentiert. Da die im Rahmen dieser Arbeit betrachteten Modellierungsschritte auf Basis der allgemeinen Definitionen der parametrischen Modellierung aufgebaut wurden, repräsentiert der erzeugte Graph das parametrische Modell unabhängig von einer proprietären CAD-Anwendung.

Damit wird die Möglichkeit geschaffen, verschiedene und auch konkurrierende Modellierungsschritte, -abläufe und prozesse formal zu beschreiben und in einem Entwurfsprozess anzuwenden, der als Ergebnis ein parametrisiertes Modell erzeugt, das nachträglich vom Nutzer durch die Änderung der Parameterwerte verändert werden kann. Damit wird eine schnelle, nutzergesteuerte Exploration von verschiedenen Entwürfen möglich und es ergibt sich das Potenzial, den Entwurfsprozess zu beschleunigen bzw. die umfangreiche Untersuchung von Alternativen erst zu ermöglichen. Gerade bei mehrskaligen Modellen, deren manuelle Erstellung nach Borrmann *et al.* (2014) aufwändig und fehleranfällig ist, birgt diese Teilautomatisierung, bei der lediglich die gewünschten Regeln vom Nutzer ausgewählt werden müssen, eine potenziell große Arbeitserleichterung.

Es ist allerdings zu beachten, dass der Aufwand zur Erstellung der Graphersetzungsregeln mit einem nicht unerheblichen Aufwand verbunden ist. Neben der erforderlichen Einarbeitungzeit in den vorgestellten Ansatz und die grundlegenden Mechanismen der Graphersetzung, muss zusätzlich berücksichtigt werden, dass ein tiefes Verständnis der Zusammenhänge parametrischer Modelle notwendig ist. Auch können die praktischen (aber meist proprietären) Hilfsfunktionen, die aktuelle parametrische CAD-Anwendungen bieten, nicht so genutzt werden, wie es bei einer manuellen Modellierung in der CAD-Anwendung möglich wäre. Hier können beispielsweise parametrisierte Rechtecke mit einer entsprechenden Funktion erzeugt werden oder die Anwendung erstellt schon während der Modellierung parametrische Zwangsbedingungen, die der vermutlichen (oder vermeintlichen) Nutzerabsicht entsprechen. Bei der Formalisierung von Modellierungsoperationen in einer Graphersetzungsregel müssen *alle* parametrischen Zusammenhänge konkret definiert werden. Weiterhin müssen viele Informationen, die bei einer manuellen Modellierung ganz intuitiv durch den Nutzer auf Basis der durchgängigen visuellen Darstellung des Modells festgelegt werden, explizit definiert werden. Hier ist als Beispiel zu nennen, dass es bei einer manuellen Modellierung in der Regel keine Rolle spielt, ob der Anfangs- oder Endpunkt einer Linie mit einem anderen geometrischen Element parametrisch verknüpft wird, da durch den visuellen Kontext intuitiv der gewünschte Punkt ausgewählt wird.

Die Notwendigkeit der vollständig expliziten Definition all dieser Informationen in der graphbasierten Repräsentation birgt neben einem zusätzlichen Aufwand allerdings den großen Vorteil, dass die parametrische Funktionalität des Modells vollständig durch den Nutzer nachvollzogen werden kann und muss – was eine wesentliche Grundlage für eine Automatisierung darstellt. Während der Erstellung der in dieser Arbeit aufgeführten Beispiele ist aufgefallen, dass die so erstellten Modelle deutlich nachvollziehbarer aufgebaut wurden, da es nicht möglich war „Abkürzungen“ in Form von vordefinierten Funktionen oder automatisch erstellten parametrischen Zwangsbedingungen zu verwenden. Letztlich ist die Folge der Notwendigkeit einer vollständigen expliziten Definition der parametrischen Modellierungsoperationen und damit auch des parametrischen Modells die vollständige Hoheit des Regelerstellers über die Funktionalität einer Regel. Gerade wenn eine Regel nach der Erstellung vielfach automatisiert angewendet werden soll, ist dies als Vorteil zu werten, da der Nutzer die vollständige Kontrolle über seinen Entwurf behält.

Somit muss im Einzelfall analysiert werden, wann der Aufwand für eine teilautomatisierte Umsetzung von Modellierungsoperationen mit Hilfe eines Graphersetzungssystems gerechtfertigt ist. Hier ist zwischen dem Aufwand zur Automatisierung und dem erwarteten Nutzen abzuwägen, wobei sich der initiale Aufwand potenziell auf lange Sicht reduzieren lässt, sobald eine größere Anzahl von Graphersetzungsregeln definiert wurde, die dann vergleichsweise einfach für die Anwendung in verschiedenen Kontexten angepasst werden können. Da es sich bei der Analyse von initialem Aufwand und potenziellem Nutzen um eine grundlegende Fragestellung, die bei jeglicher Automatisierung von Prozessen betrachtet werden muss, handelt, sollte diese Thematik nicht als maßgebliche Einschränkung des vorgestellten Ansatzes betrachtet werden.

Abschließend ist festzuhalten, dass der hier vorgestellte Ansatz nur einen Teil des Spektrums an Funktionalitäten, die parametrische CAD-Anwendungen bieten, beinhaltet – wenngleich die wesentlichen und grundlegenden Prinzipien der parametrischen Modellierung betrachtet werden. Insbesondere bei der Verwendung von impliziten Modellierungsoperationen und der dadurch erstellten impliziten Geometrie (siehe Abschnitt 6.2.6) bestehen Einschränkungen bei der weiteren Verwendung. Beispielsweise können in gängigen CAD-Anwendungen die erstellten Volumenkörper als Basis für die Projektion von Geometrie in eine Skizze genutzt werden. Diese Funktionalität wurde bei der Konzeption des hier vorgestellten Graphersetzungssystems nicht berücksichtigt, wäre aber in einer erweiterten Version umsetzbar.

Hierbei ist auch anzumerken, dass das vorrangige Ziel der hier vorgestellten Methode darin besteht, Modellierungsoperationen zu beschreiben und dadurch parametrische Modelle zu erzeugen. Teile des Modells, die erst durch die Ausführung der formalisierten Modellierungsoperationen in einer CAD-Anwendung entstehen, können bei Berücksichtigung der genannten Zielsetzung nicht vollumfänglich im Graphen abgebildet werden. Der manuelle Prozess der Modellerstellung basiert auch darauf, dass durch die verwendete

CAD-Anwendung kontinuierlich das aktuelle Ergebnis der vom Nutzer durchgeführten Modellierungsschritte erstellt und visualisiert wird. So können weitere Operationen auf Basis der bereits erstellten Geometrie aufgebaut werden. Dies setzt aber voraus, dass die Modellierungsoperationen und der Modellierkern des CAD-Systems aufeinander abgestimmt sind. Eine per Graphersetzungsregel definierte Modellierungsoperation kann immer nur auf einem Modell aufbauen und auf ein Modell angewendet werden, das mittels eines Graphen repräsentiert ist. Um dies für das vollständige Resultat insbesondere von impliziten Modellierungsoperationen umzusetzen, müsste die in der CAD-Anwendung erzeugte Geometrie vollständig ausgelesen und in die graphbasierte Repräsentation zurückgeführt werden, was wiederum bedeutet, dass sich die graphbasierte Repräsentation stark an die Funktionen des Geometrie-Kernes der CAD-Anwendung anpassen müsste. Denkbar wäre es stattdessen, Teile des Mustergraphen einer Ersetzungsregel über eine manuelle Auswahl durch den Nutzer zu bestimmen. Auch dies stellt eine mögliche Erweiterung des Ansatzes dar.

Trotz der genannten Einschränkungen kann durch die im Rahmen dieser Arbeit erzielten Erkenntnisse festgehalten werden, dass die entwickelte Methode einen grundlegenden Beitrag zur perspektivischen Automatisierung von parametrischer Modellierung im Kontext von Entwurf und Erstellung von Bauwerken bzw. deren modellbasierter Repräsentation liefert. Dies ist auch deswegen der Fall, da die komplexen Zusammenhänge eines parametrischen Modells durch die Anwendung dieser Methode im Detail explizit formalisiert werden müssen, was zwangsläufig zu einem tiefen Verständnis dieser Zusammenhänge durch den Ersteller der Regeln führt. Durch dieses Verständnis ist auch sichergestellt, dass die damit mögliche Automatisierung von Modellierungs- und Entwurfsvorgängen auf einer nachvollziehbaren und formal beschriebenen Basis beruht. Somit können die formalisierten Modellierungsoperationen wiederholt und automatisiert angewendet werden, wobei sichergestellt ist, dass die erzeugten Modelle der Intention und den Qualitätsansprüchen des Erstellers der Regeln entsprechen. Weiterhin wird das in den Modellierungsoperationen enthalten Ingenieurwissen zur Erstellung parametrische Modelle für bestimmte Anwendungsfälle durch die graphbasierte Formalisierung externalisiert und ist unabhängig von einer bestimmten (proprietären) CAD-Umgebung verwendbar. Dieses Wissen kann somit in andere CAD-Umgebungen übertragen, herstellerneutral dokumentiert und an Dritte weitergegeben werden.

8.2 Ausblick

Wie schon im vorangegangenen Abschnitt angedeutet, sind verschiedene Möglichkeiten zur Weiterentwicklung des in dieser Arbeit vorgestellten Ansatzes denkbar.

Dies betrifft einerseits die Erweiterung der durch das Graphersetzungssystem abgebildeten Möglichkeiten zur parametrischen Modellierung. Auch wenn die grundlegenden Funktionen, die das Konzept der parametrischen Modellierung bietet und die in kommerziellen parame-

trischen CAD-Anwendungen implementiert sind, in dieser Arbeit aufgegriffen wurden, so gibt es doch verschiedene weitere Möglichkeiten, die betrachtet werden können. Dies betrifft beispielsweise sogenannte adaptive Bauteile (siehe auch Kapitel 3), deren Parameterwerte sich durch die Ausrichtung dieser Bauteile innerhalb einer Baugruppe automatisch anpassen und so ihre Abmessung entsprechend anderer Bauteile der Baugruppe verändern. Eine entsprechende Erweiterung der Definition des Graphersetzungssystems ist grundsätzlich denkbar. Diese müssten aber mit Änderungen an den bestehenden Definitionen einhergehen. Dies betrifft beispielsweise die Repräsentation von dimensionalen Zwangsbedingungen, die mittels zusätzlicher Attribute als adaptiv gekennzeichnet werden müssen. Weiterhin wäre zu untersuchen, wie eine entsprechende Repräsentation mittels der API einer parametrischen CAD-Anwendung umgesetzt werden kann. Hierbei muss in jedem Fall berücksichtigt werden, dass die graphbasierte Repräsentation allgemeingültig bleibt und sich an den Grundlagen der parametrischen Modellierung orientiert.

Eine weitere Möglichkeit besteht darin, die Visualisierung eines Modells in einer CAD-Anwendung zu nutzen, um formalisierte Modellierungsoperationen durch Nutzereingaben direkt am Modell zu unterstützen. So könnten Graphersetzungsregeln auch auf Modelle angewendet werden, die manuell erstellt wurden. Damit kann beispielsweise ein Modell in einem niedrigen Detaillierungsgrad durch den Nutzer erstellt und anschließend durch Graphersetzungsregeln detailliert werden. Dass dies grundsätzlich möglich ist, wurde bereits anhand der Detaillierung von Trägern untersucht. Dabei wurde das als Stab abstrahierte Modell eines Trägers durch den Nutzer modelliert. Anschließend wurde das geometrische Modell mittels der durch eine Graphersetzungsregel beschriebenen Modellierungsoperation verfeinert. So können letztlich die Stärken der durch die Visualisierung des Modells unterstützten manuellen Modellierung mit den Vorteilen einer formalen Definition von Modellierungsoperationen kombiniert werden.

Perspektivisch kann auch untersucht werden, ob es möglich ist, die Abfolge der Schritte einer manuellen Modellierung in einer CAD-Anwendung automatisiert auszulesen und in eine graphbasierte Repräsentation zu überführen. Die so identifizierten Modellierungsoperationen können anschließend genutzt werden, um entsprechende Graphersetzungsregeln teilweise automatisiert zu erstellen und anschließend gegebenenfalls manuell anzupassen. Dies würde den Aufwand bei der Erstellung von Graphersetzungsregeln deutlich reduzieren. Allerdings ist dies in kommerziellen CAD-Anwendungen nur dann umsetzbar, wenn es die jeweilige API ermöglicht, die Eingaben des Nutzers und die Beschaffenheit eines Modells vollständig auszulesen. Darauf aufbauend muss untersucht werden, ob und wie die so gesammelten Informationen tatsächlich in eine graphbasierte Repräsentation überführt werden können. Kann dies jedoch umgesetzt werden, ist es perspektivisch auch denkbar die graphbasierte Repräsentation und die Veränderung von parametrischen Modellen im Kontext von Machine Learning und Graph Neural Networks zu untersuchen. Beispielsweise könnte

hier die graphbasierte Repräsentation des Ergebnisses der vom Nutzer durchgeführten Modellierungsoperationen durch ein Graph Neural Network analysiert werden.

In jedem Fall bietet der Grundgedanke, parametrische Modellierungsoperationen formal und herstellerneutral zu definieren eine Vielzahl an Anwendungsmöglichkeiten, um die Erstellung von Bauwerksmodellen während eines Entwurfsprozesses zu vereinfachen und zu beschleunigen. Da die entsprechenden Modelle eine wesentliche Grundlage für die Anwendung von digitalen Methoden im Bauwesen sind, erschließt sich so ein großes Forschungspotenzial.

Anhang A

Definition des Graph-Metamodells in GrGen.NET

Im folgenden Listing A.1 ist die vollständige Definition des Graph-Metamodells in der *Graph Model Language* von GrGen.NET dargestellt.

```
1  // Basics
2  abstract node class BasicNode{
3      name:string;
4  }
5  abstract directed edge class basicEdge;
6
7  //------------------------------------------------------------------------
8  // Assembly
9  // Nodes
10 abstract node class AssemblyNode extends BasicNode;
11 node class Assembly extends AssemblyNode;
12 node class Part extends AssemblyNode;
13
14 // Edges
15 abstract directed edge class assemblyEdge extends basicEdge;
16 directed edge class containPart extends assemblyEdge
17     connect Assembly[1:*] --> Part[0:1]
18 {
19     fixed: boolean = false;
20     x:double = 0;
21     y:double = 0;
22     z:double = 0;
```

```
}
abstract directed edge class mConstraint extends assemblyEdge
    connect DerivedGeo[*] ?--? DerivedGeo[*];
directed edge class mCoincident extends mConstraint
    connect ProjectedPoint[*] ?--? ProjectedPoint[*], copy extends;
directed edge class mCollinear extends mConstraint
    connect ProjectedLine[*] ?--? ProjectedLine[*], copy extends
{
    direction:boolean = false;
}
directed edge class mCoplanar extends mConstraint
    connect copy extends
{
    normal:boolean = false;
}

//------------------------------------------------------------------------
//Part
// Enumerations
enum WpType {xy, xz, zy}
enum PathType {linear, interpolate}

// Nodes
abstract node class PartNode extends BasicNode;

    node class Sketch extends PartNode;
    node class Sketch3D extends PartNode{
        type:PathType = PathType::linear;
    }
    node class Point3D extends PartNode{
        x:double = 0;
        y:double = 0;
        z:double = 0;
    }
    node class Workplane extends PartNode{
        type:WpType;
    }
    node class DerivedGeo extends PartNode;

    abstract node class SweepingOp extends PartNode;
        node class Extrusion extends SweepingOp{
            direction:boolean;
            length:double;
        }
        node class Sweep extends SweepingOp;

    abstract node class BooleanOp extends PartNode;
        node class Union extends BooleanOp;
        node class Difference extends BooleanOp;
```

```
72
73      abstract node class SemanticNode extends PartNode;
74          node class TextualAtt extends SemanticNode{
75              value:string;
76          }
77          node class NumericalAtt extends SemanticNode{
78              value:double;
79          }
80
81  // Edges
82  abstract directed edge class partEdge extends basicEdge;
83      directed edge class contain extends partEdge
84          connect Part[*] --> Workplane[1],
85          Part[*] --> Sketch3D[1],
86          Sketch[+] --> SketchNode[1],
87          Workplane[+] --> Sketch[1],
88          Sketch3D[+] --> Point3D[1];
89      directed edge class project extends partEdge
90          connect Point[*] --> ProjectedPoint[0:1],
91          Line[*] --> ProjectedLine[0:1],
92          Circle[*] --> ProjectedCircle[0:1],
93          Arc[*] --> ProjectedArc[0:1];
94      directed edge class depend extends partEdge
95          connect Point3D[*] --> Point3D[*],
96          Point3D[*] --> Workplane[0:1],
97          DerivedGeo[*] --> Workplane[0:1],
98          Sketch3D[*] --> Sweep[1],
99          Sketch[*] --> SweepingOp[1],
100         SweepingOp[*] --> DerivedGeo[1],
101         Sketch[*] --> DerivedGeo[*],
102         ProjectedPoint[*] --> DerivedGeo[*],
103         ProjectedLine[*] --> DerivedGeo[*];
104     directed edge class attribute extends partEdge
105         connect Part[*] --> SemanticNode[*],
106         Assembly[*] --> SemanticNode[*],
107         SweepingOp[*] --> SemanticNode[*],
108         BooleanOp[*] --> SemanticNode[*];
109     directed edge class boolDepend extends partEdge
110         connect SweepingOp[*] --> BooleanOp[2:*]
111         {
112         substract: boolean = false;
113         }
114
115 //-------------------------------------------------------------------------
116 // Sketch
117 // Enumerations
118 enum PortType {StartPoint, EndPoint, CenterPoint, Arc, Line}
119 enum ModeType {Horizontal, Vertical, Direct}
120
```

```
121 // Nodes
122 abstract node class SketchNode extends BasicNode{
123     construction:boolean = false;
124 }
125     node class ProjectedPoint extends SketchNode;
126         node class Point extends ProjectedPoint{
127             x:double = 0;
128             y:double = 0;
129         }
130     node class ProjectedLine extends SketchNode;
131         node class Line extends ProjectedLine{
132             x_s:double = 0;
133             y_s:double = 0;
134             x_e:double = 0;
135             y_e:double = 0;
136         }
137     node class ProjectedCircle extends SketchNode;
138         node class  Circle extends ProjectedCircle{
139             x:double = 0;
140             y:double = 0;
141             r:double = 1;
142         }
143     node class ProjectedArc extends SketchNode;
144         node class Arc extends ProjectedArc{
145             x_s:double = -1;
146             y_s:double = 0;
147             x_e:double = 1;
148             y_e:double = 0;
149             x_m:double = 0;
150             y_m:double = 0;
151         }
152
153 // Edges
154 abstract directed edge class portConstraint extends basicEdge{
155     port_s: PortType;
156     port_e: PortType;
157 }
158
159 abstract directed edge class sketchEdge extends basicEdge;
160     abstract directed edge class dimConstraint extends sketchEdge
161         connect SketchNode[*] ?--? SketchNode[*]
162     {
163         name:string;
164         value:double;
165         based_on:string;
166         driven: boolean = false;
167     }
168         directed edge class oneElement extends dimConstraint
169             connect copy extends;
```

```
170         directed edge class twoElement extends dimConstraint,
                portConstraint
171             connect copy extends
172         {
173             mode:ModeType;
174         }
175
176     abstract directed edge class geoConstraint extends sketchEdge;
177         directed edge class locked extends geoConstraint
178             connect SketchNode[*] ?--? SketchNode[*]
179         {
180         port: PortType;
181         }
182         directed edge class parallel extends geoConstraint
183             connect ProjectedLine[*] ?--? ProjectedLine[*];
184         directed edge class perpendicular extends geoConstraint
185             connect ProjectedLine[*] ?--? ProjectedLine[*];
186         directed edge class collinear extends geoConstraint
187             connect ProjectedLine[*] ?--? ProjectedLine[*];
188         directed edge class concentric extends geoConstraint
189             connect ProjectedCircle[*] ?--? ProjectedCircle[*];
190         directed edge class equal extends geoConstraint
191             connect ProjectedCircle[*] ?--? ProjectedCircle[*],
192             ProjectedLine[*] ?--? ProjectedLine[*];
193         directed edge class horizontal extends geoConstraint,
                portConstraint
194             connect SketchNode[*] ?--? SketchNode[*];
195         directed edge class vertical extends geoConstraint, portConstraint
196             connect SketchNode[*] ?--? SketchNode[*];
197         directed edge class coincident extends geoConstraint,
                portConstraint
198             connect SketchNode[*] ?--? SketchNode[*];
```

Listing A.1: Definition des Graph-Metamodells in GrGen.NET.

Anhang B

Weitere Graphersetzungsregeln

In den folgenden Listings sind alle Graphersetzungsregeln in der *Rule and Computations Language* des Graphersetzungs-Tools GrGen.NET vollständig definiert, die in Kapitel 7 aus Gründen der Übersichtlichkeit nicht oder nur teilweise enthalten sind.

```
1  rule LoD3_Sweep{
2      s3d:Sketch3D; if (s3d.name == "TunnelAxis");
3      s1:Sketch; if (s1.name == "AnnularGapSpaceSketch");
4      s2:Sketch; if (s2.name == "LiningSpaceSketch");
5      s3:Sketch; if (s3.name == "InteriorSpaceSketch");
6  modify{
7      sw1:Sweep;
8      sw2:Sweep;
9      sw3:Sweep;
10     s3d-:depend->sw1;
11     s3d-:depend->sw2;
12     s3d-:depend->sw3;
13     s1-:depend->sw1;
14     s2-:depend->sw2;
15     s3-:depend->sw3;
16     eval{
17         sw1.name = "AnnularGapSpace";
18         sw2.name = "LiningSpace";
19         sw3.name = "InteriorSpace";
20         }
21 }}
```

Listing B.1: Definition der Regel *LoD3_Sweep*, zur Erstellung der 3D-Volumenkörper im dritten LoD eines Schildtunnels.

```
rule LoD4_ClearanceSpace_1{
    wp:Workplane-:contain->s:Sketch;
    if (s.name == "InteriorSpaceSketch");
    s-:contain->pc0:ProjectedCircle;
    c:Circle-:project->pc0;
modify{
    wp-:contain->s1:Sketch-:contain->pc:ProjectedCircle;
    c-:project->pc;
    s1-:contain->l:Line;
    l-:horizontal->l;
    pc-dim:twoElement->l;
    pc-co1:coincident->l;
    pc-co2:coincident->l;
    eval{
        s1.name = "ClearanceSpaceSketch";
        pc.construction = true;
        l.construction = true;
        l.name = "L_bottom";
        l.x_s = c.x-1; l.y_s = c.y-1;
        l.x_e = c.x+1; l.y_e = c.y-1;
        dim.port_s = PortType::CenterPoint; dim.port_s = PortType::Line;
        dim.mode = ModeType::Direct;
        co1.port_s = PortType::Arc; co1.port_s = PortType::StartPoint;
        co2.port_s = PortType::Arc; co2.port_s = PortType::EndPoint;
        }
}}
```

Listing B.2: 1. Regel im Modellierungsprozess zur Erstellung des *ClearanceSpace* im LoD 4 eines Schildtunnels.

```
rule LoD4_ClearanceSpace_2{
    s:Sketch;
    if (s.name == "ClearanceSpaceSketch");
    s-:contain->pc:ProjectedCircle;
    s-:contain->lb:Line;
modify{
    lr:Line; ll:Line; lt:Line;
    lr-:vertical->lr; ll-:vertical->ll;
    lt-:horizontal->lt;
    pc-co1:coincident->ll; pc-co2:coincident->lr;
    lb-co3:coincident->ll; lb-co4:coincident->lr;
    ll-co5:coincident->lt; lr-co6:coincident->lt;
    lr-dim:twoElement->ll;
    s-:contain->ll;
    s-:contain->lr;
    s-:contain->lt;

    eval{
```

```
        ll.construction = true;
        lr.construction = true;
        lt.construction = true;
        def var mx:double = (lb.x_s + lb.x_e) / 2;
        ll.x_s = mx - 1; ll.y_s = lb.y_s;
        ll.x_e = mx - 1; ll.y_e = lb.y_s + 1;
        ll.name = "L_left";
        lr.x_s = mx + 1; lr.y_s = lb.y_s + 1;
        lr.x_e = mx + 1; lr.y_e = lb.y_s + 1;
        lr.name = "L_right";
        lt.x_s = ll.x_e; lt.y_s = lb.y_s + 1;
        lt.x_e = lr.x_e; lt.y_e = lb.y_s + 1;
        lt.name = "L_top";
        co1.port_s = PortType::Arc; co1.port_s = PortType::EndPoint;
        co2.port_s = PortType::Arc; co2.port_s = PortType::EndPoint;
        co3.port_s = PortType::Line; co3.port_s = PortType::StartPoint;
        co4.port_s = PortType::Line; co4.port_s = PortType::StartPoint;
        co5.port_s = PortType::EndPoint; co5.port_s = PortType::StartPoint;
        co6.port_s = PortType::EndPoint; co6.port_s = PortType::EndPoint;
        dim.port_s = PortType::Line; dim.port_s = PortType::Line;
        dim.name = "ClearanceWidth";
        dim.mode = ModeType::Direct;
        dim.value = 4.0;
        }
}}
```

Listing B.3: 2. Regel im Modellierungsprozess zur Erstellung des *ClearanceSpace* im LoD 4 eines Schildtunnels.

```
rule LoD4_ClearanceSpace_3{
        s:Sketch; if (s.name == "ClearanceSpaceSketch");
        s-:contain->l_b:Line; if (l_b.name == "L_bottom");
        s-:contain->l_t:Line; if (l_t.name == "L_top");
        s-:contain->l_l:Line; if (l_l.name == "L_left");
        s-:contain->l_r:Line; if (l_r.name == "L_right");
modify{
    l:Line; r:Line; s-:contain->l; s-:contain->r;
    b:Line; t:Line; s-:contain->b; s-:contain->t;
    lt:Line; rt:Line; s-:contain->lt; s-:contain->rt;
    bl:Line; br:Line; s-:contain->bl; s-:contain->br;
    b-co1:coincident->bl;b-co2:coincident->br;
    bl-co3:coincident->l;br-co4:coincident->r;
    l-co5:coincident->lt;r-co6:coincident->rt;
    lt-co7:coincident->t;rt-co8:coincident->t;
    l_l-:collinear->l; l_r-:collinear->r;
    l_t-:collinear->t; l_b-:collinear->b;

    lt-dim_lt_h:twoElement->lt; rt-dim_rt_h:twoElement->rt;
    bl-dim_bl_h:twoElement->lt; br-dim_br_h:twoElement->br;
```

```
21      lt-dim_lt_v:twoElement->lt; rt-dim_rt_v:twoElement->rt;
22      bl-dim_bl_v:twoElement->lt; br-dim_br_v:twoElement->br;
23
24
25      eval{
26          def var d:double = (l_t.x_e - l_t.x_s) * 0.1;
27          l.x_s = l_b.x_s; l.y_s = l_b.y_s+d; l.x_e = l_t.x_s; l.y_e = l_t.
                y_s-d;
28          r.x_s = l_b.x_e; r.y_s = l_b.y_e+d; r.x_e = l_t.x_e; r.y_e = l_t.
                y_e-d;
29          t.x_s = l_t.x_s+d; t.y_s = l_t.y_s; t.x_e = l_t.x_e-d; t.y_e = l_t.
                y_e;
30          b.x_s = l_b.x_s+d; b.y_s = l_b.y_s; b.x_e = l_b.x_e-d; b.y_e = l_b.
                y_e;
31
32          lt.x_s = l_t.x_s; lt.y_s = l_t.y_s-d; lt.x_e = l_t.x_s+d; lt.y_e =
                l_t.y_s;
33          rt.x_s = l_t.x_e; rt.y_s = l_t.y_e-d; rt.x_e = l_t.x_e-d; rt.y_e =
                l_t.y_e;
34          br.x_s = l_b.x_e-d; br.y_s = l_b.y_e; br.x_e = l_b.x_e; br.y_e =
                l_b.y_e+d;
35          bl.x_s = l_b.x_s+d; bl.y_s = l_b.y_s; bl.x_e = l_b.x_s; bl.y_e =
                l_b.y_s+d;
36
37          co1.port_s = PortType::StartPoint; co1.port_s = PortType::
                StartPoint;
38          co2.port_s = PortType::EndPoint; co2.port_s = PortType::StartPoint;
39          co3.port_s = PortType::EndPoint; co3.port_s = PortType::StartPoint;
40          co4.port_s = PortType::EndPoint; co4.port_s = PortType::StartPoint;
41          co5.port_s = PortType::EndPoint; co5.port_s = PortType::StartPoint;
42          co6.port_s = PortType::EndPoint; co6.port_s = PortType::StartPoint;
43          co7.port_s = PortType::EndPoint; co7.port_s = PortType::StartPoint;
44          co8.port_s = PortType::EndPoint; co8.port_s = PortType::EndPoint;
45
46          dim_lt_h.port_s = PortType::StartPoint; dim_lt_h.port_s = PortType
                ::EndPoint;
47          dim_lt_h.based_on = "ClearanceWidth␣*␣0,1";
48          dim_rt_h.port_s = PortType::StartPoint; dim_rt_h.port_s = PortType
                ::EndPoint;
49          dim_rt_h.based_on = "ClearanceWidth␣*␣0,1";
50          dim_bl_h.port_s = PortType::StartPoint; dim_bl_h.port_s = PortType
                ::EndPoint;
51          dim_bl_h.based_on = "ClearanceWidth␣*␣0,1";
52          dim_br_h.port_s = PortType::StartPoint; dim_br_h.port_s = PortType
                ::EndPoint;
53          dim_br_h.based_on = "ClearanceWidth␣*␣0,1";
54          dim_lt_v.port_s = PortType::StartPoint; dim_lt_v.port_s = PortType
                ::EndPoint;
55          dim_lt_v.based_on = "ClearanceWidth␣*␣0,1";
```

```
56          dim_rt_v.port_s = PortType::StartPoint; dim_rt_v.port_s = PortType
                ::EndPoint;
57          dim_lt_v.based_on = "ClearanceWidth␣*␣0,1";
58          dim_bl_v.port_s = PortType::StartPoint; dim_bl_v.port_s = PortType
                ::EndPoint;
59          dim_bl_v.based_on = "ClearanceWidth␣*␣0,1";
60          dim_br_v.port_s = PortType::StartPoint; dim_br_v.port_s = PortType
                ::EndPoint;
61          dim_br_v.based_on = "ClearanceWidth␣*␣0,1";
62          }
63  }}
```

Listing B.4: 3. Regel im Modellierungsprozess zur Erstellung des *ClearanceSpace* im LoD 4 eines Schildtunnels.

```
1   rule LoD4_ServiceSpace_1{
2       wp:Workplane-:contain->sIS:Sketch;
3       if (sIS.name == "InteriorSpaceSketch");
4       sIS-:contain->pc0:ProjectedCircle;
5       c:Circle-:project->pc0;
6       sCS:Sketch; if (sCS.name == "ClearanceSpaceSketch");
7       sCS-:contain->lb:Line; if (lb.name == "L_bottom");
8       sCS-:contain->lr:Line; if (lr.name == "L_right");
9   modify{
10      wp-:contain->s1:Sketch;
11      s1-:contain->pc:ProjectedCircle;
12      s1-:contain->plb:ProjectedLine;
13      s1-:contain->plr:ProjectedLine;
14      c-:project->pc;
15      lb-:project->plb;
16      lr-:project->plr;
17
18      eval{
19          s1.name = "ServiceSpaceSketch";
20          pc.name = "SS_circle"; pc.construction = true;
21          plb.name = "SS_bottom"; lb.construction = true;
22          plr.name = "SS_vline";
23          }
24  }}
25
26  rule LoD4_ServiceSpace_2{
27      s:Sketch; if (s.name == "ServiceSpaceSketch");
28      s-:contain->pc:ProjectedCircle; if (pc.name == "SS_circle");
29      s-:contain->plb:ProjectedLine; if (plb.name == "SS_bottom");
30      s-:contain->plr:ProjectedLine; if (plr.name == "SS_vline");
31  modify{
32      s-:contain->arc:Arc;
33      s-:contain->l:Line;
34      plr-co1:coincident->l;
```

```
35      plb-co2:coincident->l;
36      plr-co3:coincident->arc;
37      plb-co4:coincident->arc;
38      pc-:concentric->arc;
39      eval{
40          co1.port_s = PortType::StartPoint; co1.port_e = PortType::
                StartPoint;
41          co2.port_s = PortType::EndPoint; co2.port_e = PortType::EndPoint;
42          co3.port_s = PortType::EndPoint; co3.port_e = PortType::StartPoint;
43          co4.port_s = PortType::EndPoint; co4.port_e = PortType::EndPoint;
44          }
45  }}
```

Listing B.5: Die hier definierten Graphersetzungsregeln bilden den Modellierungsprozess zur Erstellung des *ServiceSpace*.

```
1   rule LoD4_FloorSpace_1{
2       wp:Workplane-:contain->sIS:Sketch;
3       if (sIS.name == "InteriorSpaceSketch");
4       sIS-:contain->pc0:ProjectedCircle;
5       c:Circle-:project->pc0;
6       sCS:Sketch; if (sCS.name == "ClearanceSpaceSketch");
7       sCS-:contain->lb:Line; if (lb.name == "L_bottom");
8       sCS-:contain->lr:Line; if (lr.name == "L_right");
9       sCS-:contain->ll:Line; if (ll.name == "L_left");
10  modify{
11      wp-:contain->s1:Sketch;
12      s1-:contain->pc:ProjectedCircle;
13      s1-:contain->plb:ProjectedLine;
14      s1-:contain->plr:ProjectedLine;
15      s1-:contain->pll:ProjectedLine;
16      c-:project->pc;
17      lb-:project->plb;
18      lr-:project->plr;
19      ll-:project->pll;
20      eval{
21          s1.name = "FloorSpaceSketch";
22          pc.name = "FS_circle"; pc.construction = true;
23          plb.name = "FS_bottom"; lb.construction = true;
24          plr.name = "FS_right";
25          pll.name = "FS_left";
26          }
27  }}
28
29  rule LoD4_FloorSpace_2{
30      s:Sketch; if (s.name == "FloorSpaceSketch");
31      s-:contain->pc:ProjectedCircle; if (pc.name == "FS_circle");
32      s-:contain->plb:ProjectedLine; if (plb.name == "FS_bottom");
33      s-:contain->plr:ProjectedLine; if (plr.name == "FS_right");
```

```
34      s-:contain->pll:ProjectedLine; if (pll.name == "FS_left");
35      plb_b:Line-:project->plb;
36      plr_b:Line-:project->plr;
37      pll_b:Line-:project->pll;
38  modify{
39      s-:contain->lhr:Line; s-:contain->lhl:Line;
40      s-:contain->lvr:Line; s-:contain->lvl:Line;
41      s-:contain->lb:Line;
42      s-:contain->arc:Arc;
43      plb-co1:coincident->lhl; plb-co2:coincident->lhr;
44      pll-co3:coincident->lhl; plr-co4:coincident->lhr;
45      pll-co5:coincident->lvl; plr-co6:coincident->lvr;
46      lvl-co7:coincident->lb; lvr-co8:coincident->lb;
47      pc-:concentric->arc;
48      plb-:parallel->lb;
49      lhl-:perpendicular->lvl;
50      lhr-:perpendicular->lvr;
51      plb-co9:coincident->arc; plb-co10:coincident->arc;
52      lvl-dim:oneElement->lvl;
53      eval{
54          dim.value = 0.8;
55          lvl.x_s = pll_b.x_s;    lvl.y_s = pll_b.y_s;
56          lvl.x_e = pll_b.x_s;    lvl.y_e = pll_b.y_s - 1;
57          lvl.name = "FL_lvl";
58          lvr.x_s = plr_b.x_s;    lvr.y_s = plr_b.y_s;
59          lvr.x_e = plr_b.x_s;    lvr.y_e = plr_b.y_s - 1;
60          lvr.name = "FL_lvr";
61          lb.x_s = pll_b.x_s; lb.y_s = pll_b.y_s - 1;
62          lb.x_e = plr_b.x_s; lb.y_e = plr_b.y_s - 1;
63          lb.name = "FL_lb";
64          co1.port_s = PortType::StartPoint; co1.port_e = PortType::
                StartPoint;
65          co2.port_s = PortType::EndPoint; co2.port_e = PortType::StartPoint;
66          co3.port_s = PortType::StartPoint; co3.port_e = PortType::EndPoint;
67          co4.port_s = PortType::StartPoint; co4.port_e = PortType::EndPoint;
68          co5.port_s = PortType::StartPoint; co5.port_e = PortType::
                StartPoint;
69          co6.port_s = PortType::StartPoint; co6.port_e = PortType::
                StartPoint;
70          co7.port_s = PortType::EndPoint; co7.port_e = PortType::StartPoint;
71          co8.port_s = PortType::EndPoint; co8.port_e = PortType::EndPoint;
72          co9.port_s = PortType::StartPoint; co9.port_e = PortType::EndPoint;
73          co10.port_s = PortType::EndPoint; co10.port_e = PortType::
                StartPoint;
74          }
75  }}
```

Listing B.6: Die hier definierten Graphersetzungsregeln bilden den Modellierungsprozess zur Erstellung des *FloorSpace*.

```
rule LoD4_TrackSpace{
    wp:Workplane-:contain->sCS:Sketch; if (sCS.name == "
        ClearanceSpaceSketch");
    sCS-:contain->plt:Line; if (plt.name == "L_bottom");
    sFS:Sketch; if (sFS.name == "FloorSpaceSketch");
    sFS-:contain->pll:Line; if (pll.name == "FL_lvl");
    sFS-:contain->plr:Line; if (plr.name == "FL_lvr");
    sFS-:contain->plb:Line; if (plb.name == "FL_lb");
modify{
    wp-:contain->s:Sketch;
    s-:contain->:ProjectedLine<-:project-plt;
    s-:contain->:ProjectedLine<-:project-pll;
    s-:contain->:ProjectedLine<-:project-plr;
    s-:contain->:ProjectedLine<-:project-plb;
    eval{
        s.name = "TrackSpaceSketch";
        }
}}
```

Listing B.7: Die hier definierte Graphersetzungsregel beschreibt den Modellierungsablauf zur Erstellung des *TrackSpace*.

```
rule LoD2_QS{
    FTS_Sketch:Sketch-:depend->FTS:Sweep<-:depend-TA:Sketch3D;
    if {FTS_Sketch.name == "FullTunnelSpaceSketch";}
    FTS_QS:Sweep;
    if {FTS_QS.name == "FullTunnelSpace_QS";}
modify{
    FTS_Sketch-:depend->FTS_temp:Sweep<-:depend-TA;
    diff:Difference; eval {diff.name = "FullTunnelSpace_QS";}
    FTS_QS-dep1:boolDepend->diff;
    FTS_temp-dep2:boolDepend->diff; eval {dep2.substract = true;}
    eval {FTS_QS.name = "";}
}}

rule LoD3_QS{
    wp:Workplane-:contain->FTS_Sketch:Sketch-:depend->FTS:Sweep;
    if {FTS_Sketch.name == "FullTunnelSpaceSketch";}
    AGS_Sketch:Sketch-:depend->AGS:Sweep;
    if {AGS_Sketch.name == "AnnularGapSpaceSketch";}
    LS_Sketch:Sketch-:depend->LS:Sweep<-:depend-TA:Sketch3D;
    if {LS_Sketch.name == "LiningSpaceSketch";}
    IS_Sketch:Sketch-:depend->IS:Sweep;
    if {IS_Sketch.name == "InteriorSpaceSketch";}
    AGS_Sketch-:contain->c1:Circle; if {c1.name == "AGS_innerCircle";}
    AGS_Sketch_QS:Sketch-:depend->AGS_QS:Sweep;
    if {AGS_Sketch_QS.name == "AnnularGapSpaceSketch_QS";}
    LS_Sketch_QS:Sketch-:depend->LS_QS:Sweep;
```

```
27      if {LS_Sketch_QS.name == "LiningSpaceSketch_QS";}
28      IS_Sketch_QS:Sketch-:depend->IS_QS:Sweep;
29      if {IS_Sketch_QS.name == "InteriorSpaceSketch_QS";}
30      AGS_Sketch_QS-:contain->c1_QS:Circle; if {c1_QS.name == "
            AGS_innerCircle_QS";}
31      LS_Sketch_QS-:contain->c2_QS:Circle; if {c2_QS.name == "
            LS_innerCircle_QS";}
32  modify{
33      FTS_Sketch-:depend->FTS_temp:Sweep<-:depend-TA;
34      IS_Sketch-:depend->IS_temp:Sweep<-:depend-TA;
35      wp-:contain->AGS_Sketch_innerCircle:Sketch;
36      AGS_Sketch_innerCircle-:contain->ci1:Circle<-:project-c1;
37      AGS_Sketch_innerCircle-:depend->AGS_innerCircle:Sweep;
38      AGS_innerCircle<-:depend-TA;
39      wp-:contain->AGS_Sketch_QS_innerCircle:Sketch;
40      AGS_Sketch_QS_innerCircle-:contain->ci1_QS:Circle<-:project-c1_QS;
41      AGS_Sketch_QS_innerCircle-:depend->AGS_QS_innerCircle:Sweep;
42      AGS_QS_innerCircle<-:depend-TA;
43      wp-:contain->LS_Sketch_QS_innerCircle:Sketch;
44      LS_Sketch_QS_innerCircle-:contain->ci2_QS:Circle<-:project-c2_QS;
45      LS_Sketch_QS_innerCircle-:depend->LS_QS_innerCircle:Sweep;
46      LS_QS_innerCircle<-:depend-TA;
47      diff1:Difference; eval {diff1.name = "AnnularGapSpace";}
48      AGS-:boolDepend->diff1;
49      AGS_QS_innerCircle-dep1:boolDepend->diff1; eval {dep1.substract = true
            ;}
50      eval {AGS_QS.name = "";}
51      diff2:Difference; eval {diff2.name = "AnnularGapSpace_QS";}
52      AGS_QS-:boolDepend->diff2;
53      FTS_temp-dep2:boolDepend->diff2; eval {dep2.substract = true;}
54      eval {AGS_QS.name = "";}
55      diff3:Difference; eval {diff3.name = "LiningSpace";}
56      LS-:boolDepend->diff3;
57      LS_QS_innerCircle-dep3:boolDepend->diff3; eval {dep3.substract = true;}
58      eval {LS_QS.name = "";}
59      diff4:Difference; eval {diff4.name = "LiningSpace_QS";}
60      LS_QS-:boolDepend->diff4;
61      AGS_QS_innerCircle-dep4:boolDepend->diff4; eval {dep2.substract = true
            ;}
62      eval {AGS_QS.name = "";}
63      diff5:Difference; eval {diff5.name = "InteriorSpace_QS";}
64      IS_QS-:boolDepend->diff5;
65      IS_temp-dep5:boolDepend->diff5; eval {dep5.substract = true;}
66      eval {IS_QS.name = "";}
67  }}
68
69  rule LoD4_QS{
70      wp:Workplane-:contain->IS_Sketch:Sketch-:depend->IS:Sweep;
71      if {IS_Sketch.name == "InteriorSpaceSketch";}
```

```
    IS<-:depend-TA:Sketch3D;
    FS_Sketch_QS:Sketch-:depend->FS_QS:Sweep;
    if {FS_Sketch_QS.name == "FloorSpaceSketch_QS";}
    SS_Sketch_QS:Sketch-:depend->SS_QS:Sweep;
    if {SS_Sketch_QS.name == "ServiceSpaceSketch_QS";}
modify{
    IS_Sketch-:depend->IS_temp1:Sweep<-:depend-TA;
    IS_Sketch-:depend->IS_temp2:Sweep<-:depend-TA;
    diff1:Difference; eval {diff1.name = "FloorSpaceSketch_QS";}
    FS_QS-:boolDepend->diff1;
    IS_temp1-dep1:boolDepend->diff1; eval {dep1.substract = true;}
    eval {FS_QS.name = "";}
    diff2:Difference; eval {diff2.name = "ServiceSpaceSketch_QS";}
    SS_QS-:boolDepend->diff2;
    IS_temp2-dep2:boolDepend->diff2; eval {dep2.substract = true;}
    eval {SS_QS.name = SS_QS.name + "old";}
}}
```

Listing B.8: Die hier definierte Graphersetzungsregeln beschreiben die Modellierungsabläufe, mit denen die korrekte Verschneidung der Volumenkörper der *Spaces* eines Tunnels und eines Querschlags in LoD 2-4 mittels Boolescher Operationen umgesetzt wird.

```
rule LoD_1{
modify{
    p:Part;
    p-:contain->s:Sketch3D;
    s-:contain->p1:Point3D;
    s-:contain->p2:Point3D;
    s-:contain->p3:Point3D;
    s-:contain->p4:Point3D;
    s-:contain->p5:Point3D;
    s-:contain->p6:Point3D;
    p1-:depend->p2-:depend->p3-:depend->p4-:depend->p5-:depend->p6;
    p3-:attribute->tx:TextualAtt;
    eval{
        s.type = PathType::linear;
        s.name = "TunnelAxis";
        p1.x = 0.0; p1.y = 0.0; p1.z = 0.0;
        p2.x = 2.0; p2.y = 0.0; p2.z = 0.0;
        p3.x = 4.0; p3.y = 0.0; p3.z = 0.0;
        p4.x = 6.0; p4.y = 0.0; p4.z = 0.0;
        p5.x = 8.0; p5.y = 0.0; p5.z = 0.0;
        p6.x = 10.0; p6.y = 0.0; p6.z = 0.0;
        tx.name = "has_object";
        tx.value = "signal";
        }
}}

rule LoD5_WP{
```

```
28      s3d:Sketch3D;
29      p:Point3D-:attribute->tx:TextualAtt;
30      if {tx.name == "has_object"; tx.value == "signal";}
31      s3d-:contain->p;
32  modify{
33      wp:Workplane;
34      s3d-:depend->wp;
35      p-:depend->wp;
36      wp-:contain->s:Sketch;
37      eval{
38          wp.type = WpType::context;
39          s.name = "SignalSketch";
40          }
41  }}
42
43  rule LoD5_project{
44      s:Sketch; if {s.name == "SignalSketch";}
45      ss:Sketch; if (ss.name == "ServiceSpaceSketch");
46      ss-:contain->a:Arc;
47      ss-:contain->:ProjectedLine<-:project-l_cs:Line;
48      if {l_cs.name == "L_right";}
49  modify{
50      s-:contain->pa:ProjectedArc<-:project-a;
51      s-:contain->pl:ProjectedLine<-:project-l_cs;
52      eval{
53          pa.construction = true;
54          pl.construction = true;
55      }
56  }}
57
58  rule LoD5_rectangle_ex{
59      s1:Sketch; if {s1.name == "SignalSketch";}
60  modify{
61      s1-:contain->l1:Line; s1-:contain->l2:Line;
62      s1-:contain->l3:Line;s1-:contain->l4:Line;
63      l1-c1:coincident->l2; l2-c2:coincident->l3;
64      l3-c3:coincident->l4; l4-c4:coincident->l1;
65      l1-:perpendicular->l2; l1-:perpendicular->l4;
66      l1-:parallel->l3;
67      l1-dim1:oneElement->l1;
68      l2-dim2:oneElement->l2;
69      s1-:depend->ex:Extrusion;
70      eval{
71          def var s:double = 1.0;
72          l1.name = "bottom"; l2.name = "left";
73          l3.name = "top"; l4.name = "right";
74          l1.x_s = 0; l1.y_s = 0;
75          l1.x_e = 1; l1.y_e = 0;
76          l2.x_s = l1.x_s; l2.y_s = l1.y_s;
```

```
77          l2.x_e = l1.x_s; l2.y_e = l1.y_s + s;
78          l3.x_s = l1.x_s; l3.y_s = l1.y_s + s;
79          l3.x_e = l1.x_e; l3.y_e = l1.y_s + s;
80          l4.x_s = l1.x_e; l4.y_s = l1.y_e + s;
81          l4.x_e = l1.x_e; l4.y_e = l1.y_e;
82          c1.port_s=PortType::StartPoint; c1.port_e=PortType::StartPoint;
83          c2.port_s=PortType::EndPoint; c2.port_e=PortType::StartPoint;
84          c3.port_s=PortType::EndPoint; c3.port_e=PortType::StartPoint;
85          c4.port_s=PortType::EndPoint; c4.port_e=PortType::EndPoint;
86          dim1.value = 0.3;
87          dim2.value = 0.6;
88          ex.length = 0.3;
89      }
90  }}
91
92  rule LoD5_position{
93      s:Sketch; if {s.name == "SignalSketch";}
94      s-:contain->pl:ProjectedLine<-:project-;
95      s-:contain->l2:Line; if {l2.name == "left";}
96  modify{
97      pl-:collinear->l2;
98      pl-dim:twoElement->l2;
99      eval{
100         dim.value = 0.8;
101         dim.mode = ModeType::Direct;
102         dim.port_s=PortType::EndPoint; dim.port_e=PortType::EndPoint;
103     }
104 }}
```

Listing B.9: Die hier definierte Graphersetzungsregeln beschreiben die Modellierungsabläufe zur Erstellung eines Signals im LoD 5 entsprechend der Beschreibung in Abschnitt 7.2.2.

```
1   rule beam_LOD1{
2   modify{
3       p:Part-:contain->wp:Workplane;
4       eval {wp.type = WpType::xy;}
5       wp-:contain->s1:Sketch; eval {s1.name = "IBeamS";}
6       s1-:contain->l1:Line; s1-:contain->l2:Line;
7       s1-:contain->l3:Line;s1-:contain->l4:Line;
8       l1-c1:coincident->l2; l2-c2:coincident->l3;
9       l3-c3:coincident->l4; l4-c4:coincident->l1;
10      l1-:horizontal->l1; l3-:horizontal->l3;
11      l2-:vertical->l2; l4-:vertical->l4;
12      l1-dim1:oneElement->l1;
13      l2-dim2:oneElement->l4;
14      s1-:depend->ex:Extrusion;
15      eval{
16          def var b:double = 6.0;
17          def var h:double = 10.0;
```

```
18          l1.name = "U"; l2.name = "L";
19          l3.name = "O"; l4.name = "R";
20          l1.x_s = 0; l1.y_s = 0; l1.x_e = b; l1.y_e = 0;
21          l2.x_s = 0; l2.y_s = 0; l2.x_e = 0; l2.y_e = h;
22          l3.x_s = 0; l3.y_s = h; l3.x_e = b; l3.y_e = h;
23          l4.x_s = b; l4.y_s = h; l4.x_e = b; l4.y_e = 0;
24          c1.port_s=PortType::StartPoint; c1.port_e=PortType::StartPoint;
25          c2.port_s=PortType::EndPoint; c2.port_e=PortType::StartPoint;
26          c3.port_s=PortType::EndPoint; c3.port_e=PortType::StartPoint;
27          c4.port_s=PortType::EndPoint; c4.port_e=PortType::EndPoint;
28          dim1.value = b; dim1.name = "dim_b";
29          dim2.value = h; dim2.name = "dim_h";
30          ex.length = 100;
31      }
32  }}
33
34  rule beam_LOD2{
35      sk:Sketch; if (sk.name == "IBeamS");
36      sk-:contain->l1:Line; if (l1.name == "U");
37      sk-:contain->l2:Line; if (l2.name == "L");
38      sk-:contain->l3:Line; if (l3.name == "O");
39      sk-:contain->l4:Line; if (l4.name == "R");
40      l1-dimA:oneElement->l1; l2-dimB:oneElement->l4;
41      replace{
42      sk-:contain->sL:Line; sk-:contain->sR:Line;
43      sk-:contain->fo1:Line; sk-:contain->fo2:Line;
44      sk-:contain->foL:Line; sk-:contain->foR:Line;
45      sk-:contain->fo:Line; eval{fo.name = "Flansch_LO";}
46      sk-:contain->fu1:Line; sk-:contain->fu2:Line;
47      sk-:contain->fuL:Line; sk-:contain->fuR:Line;
48      sk-:contain->fu:Line; eval{fu.name = "Flansch_LU";}
49      sk-:contain->p:Point; eval{p.name = "Flasnch_MP";}
50
51      sL-co1:coincident->fo2;
52      eval{co1.port_s = PortType::EndPoint; co1.port_e = PortType::StartPoint
            ;}
53      fo2-co2:coincident->foL;
54      eval{co2.port_s = PortType::EndPoint; co2.port_e = PortType::StartPoint
            ;}
55      foL-co3:coincident->fo;
56      eval{co3.port_s = PortType::EndPoint; co3.port_e = PortType::StartPoint
            ;}
57      fo-co4:coincident->foR;
58      eval{co4.port_s = PortType::EndPoint; co4.port_e = PortType::StartPoint
            ;}
59      foR-co5:coincident->fo1;
60      eval{co5.port_s = PortType::EndPoint; co5.port_e = PortType::StartPoint
            ;}
61      fo1-co6:coincident->sR;
```

```
62      eval{co6.port_s = PortType::EndPoint; co6.port_e = PortType::StartPoint
            ;}
63      sR-co7:coincident->fu1;
64      eval{co7.port_s = PortType::EndPoint; co7.port_e = PortType::StartPoint
            ;}
65      fu1-co8:coincident->fuR;
66      eval{co8.port_s = PortType::EndPoint; co8.port_e = PortType::StartPoint
            ;}
67      fuR-co9:coincident->fu;
68      eval{co9.port_s = PortType::EndPoint; co9.port_e = PortType::StartPoint
            ;}
69      fu-co10:coincident->fuL;
70      eval{co10.port_s = PortType::EndPoint; co10.port_e = PortType::
            StartPoint;}
71      fuL-co11:coincident->fu2;
72      eval{co11.port_s = PortType::EndPoint; co11.port_e = PortType::
            StartPoint;}
73      fu2-co12:coincident->sL;
74      eval{co12.port_s = PortType::EndPoint; co12.port_e = PortType::
            StartPoint;}
75      fo-co13:coincident->p;
76      eval{co13.port_s = PortType::CenterPoint; co13.port_e = PortType::none
            ;}
77      fo-dim_b:oneElement->fo;
78      foL-dim_t:oneElement->foL;
79      sL-dim_s:twoElement->sR;
80      eval{dim_s.port_s = PortType::Line; dim_s.port_e = PortType::Line;}
81      fo-dim_h:twoElement->fu;
82      eval{dim_h.port_s = PortType::Line; dim_h.port_e = PortType::Line;}
83      fo-:horizontal->fo; fo1-:horizontal->fo1; fo2-:horizontal->fo2;
84      fu1-:horizontal->fu1; fu2-:horizontal->fu2;
85      sL-:vertical->sL; sR-:vertical->sR;
86      foL-:vertical->foL; foR-:vertical->foR;
87      fuL-:vertical->fuL; fuR-:vertical->fuR;
88      foL-:equal->fuR;
89      fo2-:equal->fo1; fo2-:equal->fu2; fo2-:equal->fu1;
90      fo1-:collinear->fo2; fu1-:collinear->fu2;
91
92      eval {
93          def var b:double = dimA.value;
94          def var h:double = dimB.value;
95          def var t:double = 1; def var s:double = 1;
96          dim_b.name = "dim_b"; dim_b.value = b;
97          dim_t.name = "dim_t"; dim_t.value = t;
98          dim_s.name = "dim_s"; dim_s.value = s;
99          dim_h.name = "dim_h"; dim_h.value = h;
100         sL.x_s = (b-s)/s; sL.y_s = t;
101         sL.x_e = (b-s)/s; sL.y_e = h-t;
102         sR.x_s = (b+s)/2; sR.y_s = h-t;
```

```
103         sR.x_e = (b+s)/2; sR.y_e = t;
104         fo1.x_s = b; fo1.y_s = h-t;
105         fo1.x_e = (b+s)/2; fo1.y_e = h-t;
106         fo2.x_s = (b-s)/2; fo2.y_s = h-t;
107         fo2.x_e = 0; fo2.y_e = h-t;
108         foL.x_s = 0; foL.y_s = h-t;
109         foL.x_e = 0; foL.y_e = h;
110         foR.x_s = b; foR.y_s = h;
111         foR.x_e = b; foR.y_e = h-t;
112         fo.x_s = 0; fo.y_s = h;
113         fo.x_e = b; fo.y_e = h;
114         fu1.x_s = (b+s)/2; fu1.y_s = t;
115         fu1.x_e = b; fu1.y_e = t;
116         fu2.x_s = 0; fu2.y_s = t;
117         fu2.x_e = (b-s)/2; fu2.y_e = t;
118         fuL.x_s = 0; fuL.y_s = 0;
119         fuL.x_e = 0; fuL.y_e = t;
120         fuR.x_s = b; fuR.y_s = t;
121         fuR.x_e = b; fuR.y_e = 0;
122         fu.x_s = b; fu.y_s = 0;
123         fu.x_e = 0; fu.y_e = 0;
124         }
125     }
126 }
```

Listing B.10: Die hier definierte Graphersetzungsregeln beschreiben die Modellierungsabläufe zur Detaillierung eines Profilträgers.

Literaturverzeichnis

Abualdenien, J. & Borrmann, A. (2019). A meta-model approach for formal specification and consistent management of multi-LOD building models. *Advanced Engineering Informatics*, *40*(April), 135–153. https://doi.org/10.1016/j.aei.2019.04.003

Abualdenien, J., Borrmann, A. & König, M. (2021). Ausarbeitungsgrade von BIM-Modellen. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (2. Aufl.). Springer Vieweg.

Abulawi, J. (2012). *Ansatz zur Beherrschung der Komplexität von vernetzten 3D-CAD-Modellen* (Dissertation). Universität der Bundeswehr Hamburg.

Agugiaro, G. (2016). Energy planning tools and CityGML-based 3D virtual city models: experiences from Trento (Italy). *Applied Geomatics*, *8*(1), 41–56. https://doi.org/10.1007/s12518-015-0163-2

Aldefeld, B. (1988). Variation of geometries based on a geometric-reasoning method. *Computer-Aided Design*, *20*(3), 117–126. https://doi.org/10.1016/0010-4485(88)90019-X

Aliaga, D., Cohen, J., Wilson, A., Baker, E., Zhang, H., Erikson, C., Hoff, K., Hudson, T., Stuerzlinger, W., Bastos, R., Whitton, M., Brooks, F. & Manocha, D. (1999a). MMR: An interactive massive model rendering system using geometric and image-based acceleration. *Proceedings of the Symposium on Interactive 3D Graphics*, 199–206.

Aliaga, D. G. & Lastra, A. (1999b). Automatic image placement to provide a guaranteed frame rate. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, 307–316. https://doi.org/10.1145/311535.311574

Alves, S., Fernández, M. & Mackie, I. (2011). A new graphical calculus of proofs. *Electronic Proceedings in Theoretical Computer Science*, *48*, 69–84. https://doi.org/10.4204/eptcs.48.8

Amadori, K. (2012). *Geometry Based Design Automation – Applied to Aircraft Modelling and Optimization* (Dissertation). Linköping University.

Amadori, K., Tarkian, M., Ölvander, J. & Krus, P. (2012). Flexible and robust CAD models for design automation. *Advanced Engineering Informatics*, *26*(2), 180–195. https://doi.org/10.1016/j.aei.2012.01.004

Amann, J., Borrmann, A., Hegemann, F., Jubierre, J. R., Flurl, M., Koch, C. & König, M. (2013). A refined product model for shield tunnels based on a generalized approach for alignment representation. *Proc. of the ICCBEI 2013*.

Amor, R. (2015). Analysis of the Evolving IFC Schema. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*, 39–48.

Anantha, R., Kramer, G. A. & Crawford, R. H. (1996). Assembly modelling by geometric constraint satisfaction. *Computer-Aided Design*, *28*(9), 707–722. https://doi.org/https://doi.org/10.1016/0010-4485(96)00001-2

Andrei, O. & Kirchner, H. (2008). A Rewriting Calculus for Multigraphs with Ports. *Electronic Notes in Theoretical Computer Science*, *219*(100), 67–82. https://doi.org/10.1016/j.entcs.2008.10.035

Aouat, A., Bendella, F. & Deba, E. a. (2012). Tools of model transformation by graph transformation. *2012 IEEE International Conference on Computer Science and Automation Engineering*, 425–428. https://doi.org/10.1109/ICSESS.2012.6269495

Arroyo Ohori, K., Biljecki, F., Kumar, K., Ledoux, H. & Stoter, J. (2018). Modeling Cities and Landscapes in 3D with CityGML. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling* (S. 199–215). Springer International Publishing. https://doi.org/10.1007/978-3-319-92862-3_11

Babai, L. (2015). Graph Isomorphism in Quasipolynomial Time. *Proceedings of the Annual ACM Symposium on Theory of Computing*, *19-21-June*, 684–697. https://doi.org/10.1145/2897518.2897542

Bak, C. (2015). *GP 2: efficient implementation of a graph programming language* (Dissertation). University of York.

Balakrishnan, R. & Ranganathan, K. (2000). *A Textbook of Graph Theory*. Springer, New York. https://doi.org/10.2307/3620814

Balasubramanian, D., Narayanan, A., Van Buskirk, C. & Karsai, G. (2006). The graph rewriting and transformation language: GReAT. *Electronic Communications of the EASST*, *1*. https://doi.org/10.14279/tuj.eceasst.1.89.82

Banks, J. (1998). *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

Battista, G. D., Eades, P., Tamassia, R. & Tollis, I. G. (1994). Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, *4*(5), 235–282. https://doi.org/10.1016/0925-7721(94)00014-X

Belhaouari, H., Arnould, A., Le Gall, P. & Bellet, T. (2014). Jerboa: A graph transformation library for topology-based geometric modeling. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8571 LNCS*, 269–284. https://doi.org/10.1007/978-3-319-09108-2_18

Benyon, D. (1990). *Information and data modelling*. Blackwell.

Berling, R., Du, C., Hower, W. & Rosendahl, M. (1993). Modellierung geometrischer Constraints fur CAD-Anwendungen. *Tagungsband für neue Architekturkonzepte zur Gestaltung graphischer Systeme*, 19–29.

Bettig, B. & M. Hoffmann, C. (2011). Geometric Constraint Solving in Parametric Computer-Aided Design. *Journal of Computing and Information Science in Engineering*, *11*(2), 1–9. https://doi.org/10.1115/1.3593408

Bettig, B. & Shah, J. (2003). Solution Selectors: A User-Oriented Answer to the Multiple Solution Problem in Constraint Solving. *Journal of Mechanical Design*, *125*(3), 443–451. https://doi.org/10.1115/1.1587749

Bhatt, M., Borrmann, A., Amor, R. & Beetz, J. (2013). Architecture, computing, and design assistance. *Automation in Construction*, *32*, 161–164. https://doi.org/10.1016/j.autcon.2013.01.001

Bidarra, R., Nyirenda, P. J. & Bronsvoort, W. F. (2005). A feature-based solution to the persistent naming problem. *Computer-Aided Design and Applications*, *2*(1-4), 517–526.

Biljecki, F. (2017). *Level of detail in 3D city models* (Dissertation). Delft University of Technology. https://doi.org//10.4233/uuid:f12931b7-5113-47ef-bfd4-688aae3be248

Biljecki, F., Ledoux, H., Stoter, J. & Vosselman, G. (2016). The variants of an LOD of a 3D building model and their influence on spatial analyses. *ISPRS Journal of Photogrammetry and Remote Sensing*, *116*, 42–54. https://doi.org/10.1016/j.isprsjprs.2016.03.003

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S. & Çöltekin, A. (2015). Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*, *4*(4), 2842–2889. https://doi.org/10.3390/ijgi4042842

BIM2TWIN. (2021). BIM2TWIN. https://www.unismart.it/bim2twin/

BIMForum. (2020). Level of Development Specification, BIMForum USA. https://bimforum.org/lod/

Björk, B.-C. & Laakso, M. (2010). CAD standardisation in the construction industry — A process view. *Automation in Construction*, *19*(4), 398–406. https://doi.org/10.1016/j.autcon.2009.11.010

Blythe, J., McGrath, C. & Krackhardt, D. (1996). The effect of graph layout on inference from social network data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (S. 40–51). https://doi.org/10.1007/BFb0021789

Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D. & Ferro, A. (2013). A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, *14*(S7), S13. https://doi.org/10.1186/1471-2105-14-S7-S13

Boone, W. W. (1959). The word problem. *Annals of mathematics*, 207–265.

Boonstra, S. (2020). *Multi-disciplinary optimization of building spatial designs: co-evolutionary design process simulations, evolutionary algorithms, hybrid approaches* (Dissertation). Department of the Built Environment. Technische Universiteit Eindhoven.

Boonstra, S., van der Blom, K., Hofmeyer, H. & Emmerich, M. T. (2020). Conceptual structural system layouts via design response grammars and evolutionary algorithms. *Automation in Construction*, *116*(August 2020), 103009. https://doi.org/10.1016/j.autcon.2019.103009

Boonstra, S., van der Blom, K., Hofmeyer, H. & Emmerich, M. T. (2021). Hybridization of an evolutionary algorithm and simulations of co-evolutionary design processes for early-stage building spatial design optimization. *Automation in Construction*, *124*(February 2020). https://doi.org/10.1016/j.autcon.2020.103522

Borning, A. (1981). The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Languages and Systems*, *3*(4), 353–387. https://doi.org/10.1145/357146.357147

Borrmann, A. (2007). *Unterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken* (Dissertation). Technische Universität München.

Borrmann, A., Beetz, J., Koch, C., Liebich, T. & Muhic, S. (2018). Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling - Technology Foundations and Industry Practice* (S. 81–126). Springer International Publishing. https://doi.org/10.1007/978-3-319-92862-3_5

Borrmann, A., Beetz, J., Koch, C., Liebich, T. & Muhič, S. (2021a). Industry Foundation Classes - Ein herstellerunabhängiges Datenmodell für den gesamten Lebenszyklus eines Bauwerks. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 95–146). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_6

Borrmann, A. & Berkhahn, V. (2021b). Grundlagen der geometrischen Modellierung. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 35–51). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_2

Borrmann, A., Flurl, M., Jubierre, J. R., Mundani, R.-P. & Rank, E. (2014). Synchronous collaborative tunnel design based on consistency-preserving multi-scale models. *Advanced Engineering Informatics*, *28*(4), 499–517. https://doi.org/10.1016/j.aei.2014.07.005

Borrmann, A., Ji, Y., Jubierre, J. R. & Flurl, M. (2012). Procedural Modeling: A new approach to multi-scale design in infrastructure projects. *Proc. of the EG-ICE Workshop on Intelligent Computing in Civil Engineering*.

Borrmann, A. & Jubierre, J. R. (2013a). A multi-scale tunnel product model providing coherent geometry and semantics. *Proc. of the 2013 ASCE International Workshop on Computing in Civil Engineering*, 291–298. https://doi.org/10.1061/9780784413029.037

Borrmann, A., Kolbe, T., Donaubauer, A., Steuer, H., Jubierre, J. R. & Flurl, M. (2015). Multi-Scale Geometric-Semantic Modeling of Shield Tunnels for GIS and BIM Applications. *Computer-Aided Civil and Infrastructure Engineering*, *30*(4), 263–281. https://doi.org/10.1111/mice.12090

Borrmann, A., Kolbe, T. H., Donaubauer, A., Steuer, H. & Jubierre, J. R. (2013b). Transferring Multi-Scale Approaches From 3D City Modeling To Ifc-Based Tunnel Modeling. *Proc. of the 8th Int. Conf. 3D GeoInfo*.

Borrmann, A., König, M., Koch, C. & Beetz, J. (2021c). Building Information Modeling: Technologische Grundlagen und industrielle Praxis.

Borrmann, A., König, M., Koch, C. & Beetz, J. (2021d). Die BIM-Methode im Überblick. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information*

Modeling: Technologische Grundlagen und industrielle Praxis (S. 1–31). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_1

Borrmann, A. & Rank, E. (2009). Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics*, *23*(1), 32–44. https://doi.org/10.1016/j.aei.2008.06.005

Bouma, W., Fudos, I., Hoffmann, C. M., Cai, J. & Paige, R. (1995). Geometric constraint solver. *Computer-Aided Design*, *27*(6), 487–501. https://doi.org/10.1016/0010-4485(94)00013-4

Bradley, A., Li, H., Lark, R. & Dunn, S. (2016). BIM for infrastructure: An overall review and constructor perspective. *Automation in Construction*, *71*, 139–152. https://doi.org/10.1016/j.autcon.2016.08.019

Braun, A. (2020). *Automated BIM-based construction progress monitoring by processing and matching semantic and geometric data* (Dissertation). Technische Universität München.

Braun, A. & Borrmann, A. (2014). Towards automated construction progress monitoring using BIM-based point cloud processing. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*.

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 infrastructure modeling and simulation. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-4/W4*(4W4), 341–352. https://doi.org/10.5194/isprs-archives-XLII-4-W4-341-2017

Brilakis, I., Pan, Y., Borrmann, A., Mayer, H.-G., Rhein, F., Vos, C., Pettinato, E. & Wagner, S. (2020). *Report of the International Workshop on Built Environment Digital Twinning* (Techn. Ber.). Technische Universität München.

Brooks, R. & Tobias, A. (1996). Choosing the best model: Level of detail, complexity, and model performance. *Mathematical and Computer Modelling*, *24*(4), 1–14. https://doi.org/10.1016/0895-7177(96)00103-3

Brüderlin, B. D. et al. (1990). Symbolic computer geometry for computer aided geometric design. *Advances in Design and Manufacturing Systems, Tempe, AZ*.

Brüderlin, B. & Roller, D. (1998). *Geometric Constraint Solving and Applications* (B. Brüderlin & D. Roller, Hrsg.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-58898-3

Brüggemann, T. & von Both, P. (2015). 3D-Stadtmodellierung: CityGML. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling* (S. 177–192). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-05606-3_10

BuildingSMART. (2020). IFC4.3 RC2 - Release Candidate 2. https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC2/HTML/

buildingSMART. (2017). Industry Foundation Classes, Version 4.0.2.1, Documentation. https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/

Bungartz, H.-J., Griebel, M. & Zenger, C. (2002). *Einführung in die Computergraphik*. Springer Fachmedien.

Cagan, J., Campbell, M. I., Finger, S. & Tomiyama, T. (2005). A Framework for Computational Design Synthesis: Model and Applications. *Journal of Computing and Information Science in Engineering*, *5*(3), 171–181. https://doi.org/10.1115/1.2013289

Caneparo, L. (2022). Semantic knowledge in generation of 3D layouts for decision-making. *Automation in Construction*, *134*(June), 104012. https://doi.org/10.1016/j.autcon.2021.104012

Carletti, V., Foggia, P., Greco, A., Saggese, A. & Vento, M. (2020). Comparing performance of graph matching algorithms on huge graphs. *Pattern Recognition Letters*, *134*, 58–67. https://doi.org/10.1016/j.patrec.2018.06.025

Carletti, V., Foggia, P., Saggese, A. & Vento, M. (2018). Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 804–818. https://doi.org/10.1109/TPAMI.2017.2696940

Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N. V. & Wood, K. L. (2011). Computer-Based Design Synthesis Research: An Overview. *Journal of Computing and Information Science in Engineering*, *11*(2), 021003. https://doi.org/10.1115/1.3593409

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, *2*(3), 113–124.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, *2*(2), 137–167. https://doi.org/10.1016/S0019-9958(59)90362-6

Chou, S.-C. (1988). An introduction to Wu's method for mechanical theorem proving in geometry. *Journal of Automated Reasoning*, *4*(3), 237–267. https://doi.org/10.1007/BF00244942

Clark, J. H. (1976). Hierarchical geometric models for visible-surface algorithms. *Communications of the ACM*, *10*(2), 267–267. https://doi.org/10.1145/965143.563323

Cooper, D. & La Rocca, G. (2007). Knowledge-based techniques for developing engineering applications in the 21st century. *7th AIAA ATIO Conference, Belfast, Northern Ireland*.

Cordella, L., Foggia, P., Sansone, C. & Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1367–1372. https://doi.org/10.1109/TPAMI.2004.75

Corp, S. (2021). 3D ACIS Modeler. https://www.spatial.com/products/3d-acis-modeling

Dang, H. V., Tatipamula, M. & Nguyen, H. X. (2021). Cloud-based Digital Twinning for Structural Health Monitoring Using Deep Learning. *IEEE Transactions on Industrial Informatics*, 1–1. https://doi.org/10.1109/TII.2021.3115119

Daum, S. & Borrmann, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. *Advanced Engineering Informatics*, *28*(4), 272–286. https://doi.org/10.1016/j.aei.2014.06.001

Diestel, R. (1996). *Graphentheorie*. Springer.

Donaubauer, A., Esch, R. & Kolbe, T. H. (2016). Prozedurale Verfahren zur Generierung von 3D-Infrastrukturobjekten. In T. H. Kolbe, R. Bill & A. Donaubauer (Hrsg.), *Geoinformationssysteme 2016*. Wichmann.

Dori, G. (2016). *Simulation-based methods for float time determination and schedule optimization for construction projects* (Dissertation). Technische Universität München.

Eastman, C. M. (1999). *Building product models: computer environments, supporting design and construction* (1st). CRC Press, Inc.

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

Eastman, C. M., Teicholz, P., Sacks, R. & Liston, K. (2011). *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons.

Ehrig, H., Pfender, M. & Schneider, H. J. (2008). Graph-grammars: An algebraic approach, 167–180. https://doi.org/10.1109/SWAT.1973.11

Ehrig, H., Ehrig, K., Prange, U. & Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation*. Springer.

Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A. & Corradini, A. (1997). Algebraic approaches to graph transformation - Part II: Single pushout approach and comparison with double pushout approach. *Handbook Of Graph Grammars And*

Computing By Graph Transformation: Volume 1: Foundations (S. 247–312). World Scientific. https://doi.org/10.1142/9789812384720_0004

Eichhoff, J. R., Baumann, F. & Roller, D. (2016). Two approaches to the induction of graph-rewriting rules for function-based design synthesis. *Proceedings of the ASME Design Engineering Technical Conference*, *1B-2016*, 1–10. https://doi.org/10.1115/DETC2016-59915

El Saddik, A. (2018). Digital Twins: The Convergence of Multimedia Technologies. *IEEE MultiMedia*, *25*(2), 87–92. https://doi.org/10.1109/MMUL.2018.023121167

Eppstein, D. (1999). Subgraph Isomorphism in Planar Graphs and Related Problems. *Journal of Graph Algorithms and Applications*, *3*(3), 1–27. https://doi.org/10.7155/jgaa.00014

Ermel, C., Rudolf, M. & Taentzer, G. (1999). The AGG approach: Language and environment. *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 2: Applications, Languages and Tools* (S. 551–603). World Scientific.

Ershov, A. (2022). LEDAS Has Developed Five (!) Constraint Solvers. https://ledas.com/post/848-ledas-has-developed-five-constraint-solvers/

Evans, F., Skiena, S. & Varshney, A. (1996). Optimizing triangle strips for fast rendering. *Proceedings of Seventh Annual IEEE Visualization '96*, 319–326. https://doi.org/10.1109/VISUAL.1996.568125

Fan, H. & Meng, L. (2012). A three-step approach of simplifying 3D buildings modeled by CityGML. *International Journal of Geographical Information Science*, *26*(6), 1091–1107. https://doi.org/10.1080/13658816.2011.625947

Fernández, M., Kirchner, H. & Pinaud, B. (2014). Strategic port graph rewriting: An interactive modelling and analysis framework. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, *159*(Graphite), 15–29. https://doi.org/10.4204/EPTCS.159.3

Fernández, M., Pinaud, B. & Varga, J. (2020). A Port Graph Rewriting Approach to Relational Database Modelling. In M. Gabbrielli (Hrsg.), *Logic-Based Program Synthesis and Transformation* (S. 211–227). Springer International Publishing.

Franz, B. & Messner, J. (2019). Evaluating the Impact of Building Information Modeling on Project Performance. *Journal of Computing in Civil Engineering*, *33*(3), 04019015. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000832

Frappaolo, C. (2008). Implicit knowledge. *Knowledge Management Research and Practice*, *6*(1), 23–25. https://doi.org/10.1057/palgrave.kmrp.8500168

Frucht, R. (1939). Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Mathematica*, *6*, 239–250.

Fudos, I. & Hoffmann, C. M. (1997). A Graph-constructive Approach to Solving Systems of Geometric Constraints. *ACM Transactions on Graphics*, *16*(2), 179–216. https://doi.org/10.1145/248210.248223

Gan, V. J. (2022). BIM-based graph data model for automatic generative design of modular buildings. *Automation in Construction*, *134*(October 2021), 104062. https://doi.org/10.1016/j.autcon.2021.104062

Garey, M. R. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. W. H. Freeman.

Geiß, R., Batz, G. V., Grund, D., Hack, S. & Szalkowski, A. (2006). Grgen: A fast SPO-based graph rewriting tool. *Proceedings of ICGT*, *4178*, 383–397. https://doi.org/10.1.1.64.4131

Geiß, R. R. (2008). *Graphersetzung mit Anwendungen im Übersetzerbau* (Dissertation). Karlsruhe Institute of Technology.

Ghamarian, A. H., de Mol, M., Rensink, A., Zambon, E. & Zimakova, M. (2012). Modelling and analysis using GROOVE. *International Journal on Software Tools for Technology Transfer*, *14*(1), 15–40. https://doi.org/10.1007/s10009-011-0186-x

Gilbert, S. W. (1991). Model building and a definition of science. *Journal of Research in Science Teaching*, *28*(1), 73–79. https://doi.org/10.1002/tea.3660280107

Girardet, A. & Boton, C. (2021). A parametric BIM approach to foster bridge project design and analysis. *Automation in Construction*, *126*(June 2021), 103679. https://doi.org/10.1016/j.autcon.2021.103679

Grabska, E., Łachwa, A. & Ślusarczyk, G. (2012). New visual languages supporting design of multi-storey buildings. *Advanced Engineering Informatics*, *26*(4), 681–690. https://doi.org/10.1016/j.aei.2012.03.009

Grabska, E., Strug, B. & Ślusarczyk, G. (2018). A Visual Interactive Environment for Engineering Knowledge Modelling. *Proc. of EGICE 2018* (S. 219–230). Springer, Cham. https://doi.org/10.1007/978-3-319-91635-4_12

Granadeiro, V., Pina, L., Duarte, J. P., Correia, J. R. & Leal, V. M. S. (2013). A general indirect representation for optimization of generative design systems by genetic algorithms: Application to a shape grammar-based design system. *Automation in Construction*, *35*, 374–382. https://doi.org/10.1016/j.autcon.2013.05.012

Grieves, M. (2014). *Digital twin: manufacturing excellence through virtual factory replication* (Techn. Ber.). Florida Institute of Technology.

Grieves, M. & Vickers, J. (2017). Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. *Transdisciplinary Perspectives on Complex Systems* (S. 85–113). Springer International Publishing. https://doi.org/10.1007/978-3-319-38756-7_4

Gröger, G., Kolbe, T. H. & Plümer, L. (2004). Zur Konsistenz bei der Visualisierung multiskaliger 3D-Stadtmodelle. *Mitteilungen des Bundesamtes für Kartographie und Geodäsie*, *31*, 1–13.

Grohe, M. & Schweitzer, P. (2020). The graph isomorphism problem. *Communications of the ACM*, *63*(11), 128–134. https://doi.org/10.1145/3372123

Habel, A., Heckel, R. & Taentzer, G. (1996). Graph grammars with negative application conditions. *Fundamenta Informaticae*, *26*(3-4), 287–313. https://doi.org/10.3233/fi-1996-263404

Harrison, A. G. & Treagust, D. F. (2000). A typology of school science models. *International Journal of Science Education*, *22*(9), 1011–1026. https://doi.org/10.1080/095006900416884

Hartmann, D. (2000). *Objektorientierte Modellierung in Planung und Konstruktion - DFG Forschungsbericht*. Wiley-VCH.

Hartmann, T., Amor, R. & East, E. W. (2017). Information Model Purposes in Building and Facility Design. *Journal of Computing in Civil Engineering*, *31*(6), 04017054. https://doi.org/10.1061/(asce)cp.1943-5487.0000706

Hartung, R., Schönbach, R., Liepe, D. & Klemt-Albert, K. (2020). Automatized Parametric Modeling to Enhance a data-based Maintenance Process for Infrastructure Buildings. *Proceedings of the 37th International Symposium on Automation and Robotics in Construction, ISARC 2020*. https://doi.org/10.22260/ISARC2020/0038

Havemann, S. & Fellner, D. (2004). Generative parametric design of Gothic window tracery. *Proceedings Shape Modeling Applications, 2004.*, (July), 350–353. https://doi.org/10.1109/SMI.2004.1314525

He, H. & Singh, A. K. (2008). Graphs-at-a-time: Query Language and Access Methods for Graph Databases. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 405. https://doi.org/10.1145/1376616.1376660

Heckel, R. (2006). Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science*, *148*, 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

Heindorf, V. (2010). *Der Einsatz moderner Informationstechnologien in der Automobilproduktentwicklung*. Springer.

Helms, B. (2013). *Object-Oriented Graph Grammars for Computational Design Synthesis* (Dissertation). Technische Universtität München.

Helms, B., Eben, K., Shea, K. & Lindemann, U. (2009). Graph grammars - A formal method for dynamic structure transformation. *Proceedings of the 11th International DSM Conference*, 93–103.

Helms, B. & Shea, K. (2012). Computational Synthesis of Product Architectures Based on Object-Oriented Graph Grammars. *Journal of Mechanical Design*, *134*(2), 021008. https://doi.org/10.1115/1.4005592

Heok, T. K. & Daman, D. (2004). A review on level of detail. *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, 70–75. https://doi.org/10.1109/CGIV.2004.1323963

Hidalgo, M. & Joan-Arinyo, R. (2012). Computing parameter ranges in constructive geometric constraint solving: Implementation and correctness proof. *Computer-Aided Design*, *44*(7), 709–720. https://doi.org/10.1016/j.cad.2012.02.012

Hillyard, R. C. & Braid, I. C. (1978). Characterizing non-ideal shapes in terms of dimensions and tolerances. *ACM SIGGRAPH Computer Graphics*, *12*(3), 234–238. https://doi.org/10.1145/965139.807396

Hoffmann, C. M. & Joan-Arinyo, R. (2002). Kapitel 21 - Parametric Modeling. In G. Farin, J. Hoschek & M.-S. Kim (Hrsg.), *Handbook of Computer Aided Geometric Design* (S. 519–541). North-Holland. https://doi.org/https://doi.org/10.1016/B978-044451104-1/50022-8

Hoffmann, C. M. & Joan-Arinyo, R. (2005). A Brief on Constraint Solving. *Computer-Aided Design and Applications*, *2*(5), 655–663. https://doi.org/10.1080/16864360.2005.10738330

Hoisl, F. & Shea, K. (2011a). An interactive, visual approach to developing and applying parametric three-dimensional spatial grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *25*(04), 333–356. https://doi.org/10.1017/S0890060411000205

Hoisl, F. & Shea, K. (2011b). Interactive, Visual 3D Spatial Grammars. *Design Computing and Cognition '10*, 643–662. https://doi.org/10.1007/978-94-007-0510-4_34

Hoisl, F. R. (2012). *Visual , Interactive 3D Spatial Grammars in CAD for Computational Design Synthesis* (Dissertation). TU München.

Hoppe, H. (1997). View-dependent refinement of progressive meshes. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, 189–198. https://doi.org/10.1145/258734.258843

Ignatova, E., Kirschke, H., Tauscher, E. & Smarsly, K. (2015). Parametric geometric modeling in construction planning using industry foundation classes. In K. Gürlebeck & T. Lahmer (Hrsg.), *Digital Proceedings, International Conference on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering*. https://doi.org/10.25643/bauhaus-universitaet.2802

Imbach, R., Schreck, P. & Mathis, P. (2014). Leading a continuation method by geometry for solving geometric constraints. *Computer-Aided Design*.

ISO 11179. (2015). Information technology - Metadata registries (MDR) - Part 6: Registration.

Jakumeit, E., Blomer, J. & Geiß, R. (2021). The GrGen .NET User Manual. *Universität Karlsruhe, Fakultät für Informatik, Institute für Programmstrukturen und Datenorganisation, Lehrstuhl Prof. Goos*, *1*.

Jakumeit, E., Buchwald, S. & Kroll, M. (2010). GrGen.NET. https://doi.org/10.1007/s10009-010-0148-8

Ji, Y. (2014). *Durchgängige Trassen- und Brückenplanung auf Basis eines integrierten parametrischen 3D-Infrastrukturbauwerksmodells* (Dissertation). Technische Universität München.

Ji, Y., Borrmann, A., Beetz, J. & Obergrießer, M. (2013). Exchange of Parametric Bridge Models Using a Neutral Data Format. *Journal of Computing in Civil Engineering*, *27*(6), 593–606. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000286

Joan-arinyo, R. & Mata, N. (2001). Applying constructive geometric constraint solvers to geometric problems with interval parameters. *Nonlinear Analysis: Theory, Methods & Applications*, *47*(1), 213–224. https://doi.org/10.1016/S0362-546X(01)00170-5

Johannsen, G. & Alty, J. L. (1991). Knowledge engineering for industrial expert systems. *Automatica*, *27*(1), 97–114. https://doi.org/10.1016/0005-1098(91)90009-Q

Jones, D., Snider, C., Nassehi, A., Yon, J. & Hicks, B. (2020). Characterising the Digital Twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, *29*, 36–52. https://doi.org/10.1016/j.cirpj.2020.02.002

Jubierre, J. R. (2009). Analysis and coupling of a Geometric Constraint Solver with a CAD application. *Master's Thesis, Technische Universität München*.

Jubierre, J. R. (2016). *Consistency preservation methods for multi-scale design of subway infrastructure facilities* (Dissertation). Technischen Universität München.

Jubierre, J. R. & Borrmann, A. (2013). Cross-submodel consistency preservation in multi-scale engineering models. *Proc. of the 14th International Conference on Civil, Structural and Environmental Engineering Computing.*

Jubierre, J. R. & Borrmann, A. (2015). Knowledge-based engineering for infrastructure facilities: assisted design of railway tunnels based on logic models and advanced procedural geometry dependencies. *Journal of Information Technology in Constuction (ITcon)*, *20*, 421–441.

Kahani, N., Bagherzadeh, M., Cordy, J. R., Dingel, J. & Varró, D. (2019). Survey and classification of model transformation tools. *Software & Systems Modeling*, *18*(4), 2361–2397. https://doi.org/10.1007/s10270-018-0665-6

Kalkan, E., Okur, F. & Altunışık, A. (2018). Applications and usability of parametric modeling. *Journal of Construction Engineering, Management & Innovation*, *1*(3), 139–146. https://doi.org/10.31462/jcemi.2018.03139146

Kim, J., Pratt, M. J., Iyer, R. & Sriram, R. (2007). *Data exchange of parametric CAD models using ISO 10303-108* (Techn. Ber.). National Institute of Standards und Technology, Gaithersburg, USA.

Kiviniemi, A. & Fischer, M. (2009). Potential obstacles to using BIM in architectural design. In G. Shen, P. Brandon & A. Baldwin (Hrsg.), *Collaborative construction information management* (S. 36–54). Taylor; Francis.

Kniemeyer, O. (2008). *Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling* (Dissertation November 2008). BTU Cottbus.

Kolbe, T. H. (1999). *Identifikation und Rekonstruktion von Gebäuden in Luftbildern mittels unscharfer Constraints* (Dissertation). Hochschule Vechta.

Kolbe, T. H. (2009). Representing and Exchanging 3D City Models with CityGML. In J. Lee & S. Zlatanova (Hrsg.), *Proceedings of the 3rd International Workshop on 3D Geo-Information, Seoul, Korea*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-87395-2

Kolbe, T. H. & Donaubauer, A. (2021). Semantic 3D City Modeling and BIM. In W. Shi, M. F. Goodchild, M. Batty, M.-P. Kwan & A. Zhang (Hrsg.), *Urban Informatics* (S. 609–636). Springer Singapore. https://doi.org/10.1007/978-981-15-8983-6_34

Kolbe, T. H., Gröger, G. & Plümer, L. (2005). CityGML: Interoperable Access to 3D City Models. *Geo-information for Disaster Management* (S. 883–899). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-27468-5_63

Kolbeck, L., Vilgertshofer, S., Abualdenien, J. & Borrmann, A. (2021). Graph Rewriting Techniques in Engineering Design. *Frontiers in Built Environment.*

Kondyli, V., Bhatt, M. & Hartmann, T. (2018). Precedent Based Design Foundations for Parametric Design. *Advances in Computational Design*, *3*(4).

König, B., Nolte, D., Padberg, J. & Rensink, A. (2018). A Tutorial on Graph Transformation. In R. Heckel & G. Taentzer (Hrsg.), *Graph Transformation, Specifications, and Nets* (S. 83–104). Springer International Publishing. https://doi.org/10.1007/978-3-319-75396-6_5

Königseder, C. (2015). *A Methodology For Supporting Design Grammar Development And Application In Computational Design Synthesis* (Dissertation Nr. 22878).

Königseder, C. & Shea, K. (2015). Analyzing Generative Design Grammars. In J. S. Gero & S. Hanna (Hrsg.), *Design Computing and Cognition '14* (S. 363–381). Springer International Publishing. https://doi.org/10.1007/978-3-319-14956-1_21

Kraft, B. & Nagl, M. (2007a). Graphbasierte Werkzeuge zur Unterstützung des konzeptuellen Gebäudeentwurfs. In U. Rüppel (Hrsg.), *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau* (S. 155–176). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68104-5_9

Kraft, B. & Nagl, M. (2007b). Visual knowledge specification for conceptual design: Definition and tool support. *Advanced Engineering Informatics*, *21*(1), 67–83. https://doi.org/10.1016/j.aei.2006.10.001

Kraft, B. & Retkowitz, D. (2006). Graph transformations for dynamic knowledge processing. *Proceedings of the Annual Hawaii International Conference on System Sciences*, *7*. https://doi.org/10.1109/HICSS.2006.200

Kreider, R. G. & Messner, J. I. (2013). *The Uses of BIM: Classifying and Selecting BIM Uses* (Techn. Ber.). The Pennsylvania State University.

Kumar, V. (1992). Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, *13*(1), 32. https://doi.org/10.1609/aimag.v13i1.976

Laakso, M. & Kiviniemi, A. (2012). The IFC Standard - A Review of History, Development and Standardization. *ITcon Journal of Information Technology in Construction*, *17*, 134–161.

Langenhan, C., Weber, M., Liwicki, M., Petzold, F. & Dengel, A. (2013). Graph-based retrieval of building information models for supporting the early design stages. *Advanced Engineering Informatics*, *27*(4), 413–426. https://doi.org/10.1016/j.aei.2013.04.005

Latham, R. & Middleditch, A. (1996). Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design*, *28*(11), 917–928. https://doi.org/10.1016/0010-4485(96)00023-1

Leatherdale, W. H. (1974). *The Role of Analogy, Model and Metaphor in Science*. North-Holland.

Lee, G., Eastman, C. M., Sacks, R. & Navathe, S. B. (2006a). Grammatical rules for specifying information for automated product data modeling. *Advanced Engineering Informatics*, *20*(2), 155–170. https://doi.org/10.1016/j.aei.2005.08.003

Lee, G., Sacks, R. & Eastman, C. M. (2006b). Specifying parametric building object behavior (BOB) for a building information modeling system. *Automation in Construction*, *15*, 758–776. https://doi.org/10.1016/j.autcon.2005.09.009

Light, R. & Gossard, D. (1982). Modification of geometric models through variational geometry. *Computer-Aided Design*, *14*(4), 209–214. https://doi.org/10.1016/0010-4485(82)90292-5

Lu, Q., Chen, L., Li, S. & Pitt, M. (2020a). Semi-automatic geometric digital twinning for existing buildings based on images and CAD drawings. *Automation in Construction*, *115*(February). https://doi.org/10.1016/j.autcon.2020.103183

Lu, Q., Chen, L., Li, S. & Pitt, M. (2020b). Semi-automatic geometric digital twinning for existing buildings based on images and CAD drawings. *Automation in Construction*, *115*, 103183. https://doi.org/10.1016/j.autcon.2020.103183

Lu, Q., Xie, X., Parlikad, A. K., Schooling, J. M. & Konstantinou, E. (2020c). Moving from Building Information Models to Digital Twins for Operation and Maintenance. *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, 1–9. https://doi.org/10.1680/jsmic.19.00011

Lu, R. & Brilakis, I. (2019). Digital twinning of existing reinforced concrete bridges from labelled point clusters. *Automation in Construction*, *105*, 102837.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. & Huebner, R. (2003). *Level of detail for 3D graphics*. Morgan Kaufmann.

Luo, H., Li, L. & Chen, K. (2021). Parametric modeling for detailed typesetting and deviation correction in shield tunneling construction. *Automation in Construction*, (August), 104052. https://doi.org/10.1016/j.autcon.2021.104052

Machado, R., Ribeiro, L. & Heckel, R. (2015). Rule-based transformation of graph rewriting rules: Towards higher-order graph grammars. *Theoretical Computer Science*, *594*, 1–23. https://doi.org/10.1016/j.tcs.2015.01.034

Maciel, P. W. C. & Shirley, P. (1995). Visual navigation of large environments using textured clusters. *Proceedings of the 1995 symposium on Interactive 3D graphics - SI3D '95*, 95–ff. https://doi.org/10.1145/199404.199420

Mafipour, M. S., Vilgertshofer, S. & Borrmann, A. (2021). Deriving Digital Twin Models of Existing Bridges from Point Cloud Data Using Parametric Models and Metaheuristic Algorithms. *Proc. of the EG-ICE Conference 2021*.

Maher, M. L. (1990). Process models for design synthesis. *AI Magazine*, *11*(4), 49–58. https://doi.org/10.1609/aimag.v11i4.856

Marcheix, D. & Pierra, G. (2002). A survey of the persistent naming problem. *Proceedings of the seventh ACM symposium on Solid modeling and applications - SMA '02*, 13. https://doi.org/10.1145/566282.566288

Marchenko, M., Behrens, B. A., Wrobel, G., Scheffler, R. & Pleßow, M. (2011). A new method of visualization and documentation of parametric information of 3D CAD models. *Computer-Aided Design and Applications*, *8*(3), 435–448. https://doi.org/10.3722/cadaps.2011.435-448

Mata, M. P., Ahmed-Kristensen, S. & Shea, K. (2019). Implementation of Design Rules for Perception Into a Tool for Three-Dimensional Shape Generation Using a Shape Grammar and a Parametric Model. *Journal of Mechanical Design*, *141*(1), 011101. https://doi.org/10.1115/1.4040169

McCreesh, C., Prosser, P. & Trimble, J. (2020). The Glasgow Subgraph Solver: Using Constraint Programming to Tackle Hard Subgraph Isomorphism Problem Variants. In F. Gadducci & T. Kehrer (Hrsg.), *Graph Transformation* (S. 316–324). Springer International Publishing.

McKinney, K., Fischer, M. & Kunz, J. (1998). Visualization of construction planning information. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 135–138. https://doi.org/10.1145/268389.268414

Meseguer, P. (1989). Constraint satisfaction problems: An overview. *AI communications*, *2*(1), 3–17.

Migilinskas, D., Popov, V., Juocevicius, V. & Ustinovichius, L. (2013). The Benefits, Obstacles and Problems of Practical Bim Implementation. *Procedia Engineering*, *57*, 767–774. https://doi.org/10.1016/j.proeng.2013.04.097

Muenzer, C. & Shea, K. (2017). Simulation-based computational design synthesis using automated generation of simulation models from concept model graphs. *Journal of Mechanical Design, Transactions of the ASME*, *139*(7), 1–13. https://doi.org/10.1115/1.4036567

Mullins, S. & Rinderle, J. R. (1990). Grammatical Approaches to Design. *First International Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, 42–69.

Mun, D., Han, S., Kim, J. & Oh, Y. (2003). A set of standard modeling commands for the history-based parametric approach. *Computer-Aided Design*, *35*(13), 1171–1179. https://doi.org/http://dx.doi.org/10.1016/S0010-4485(03)00022-8

Münzer, C. (2016). *Constraint-Based Methods for Automated Computational Design Synthesis of Solution Spaces* (Dissertation). ETH Zürich. https://doi.org/10.3929/ETHZ-A-010603411

Münzer, C., Shea, K. & Helms, B. (2013). Automated Parametric Design Synthesis Using Graph Grammars and Constraint Solving. *Proceedings of the ASME Design Engineering Technical Conference*, *7*, 517–528. https://doi.org/10.1115/DETC2012-70313

Nagl, M., Schürr, A. & Münch, M. (2000). *Applications of Graph Transformations with Industrial Relevance* (M. Nagl, A. Schürr & M. Münch, Hrsg.; Bd. 1779). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45104-8

Neuberg, F. (2004). *Ein Softwarekonzept zur Internet-basierten Simulation des Ressourcenbedarfs von Bauwerken* (Dissertation). Technische Universität München.

Niemeijer, R., de Vries, B. & Beetz, J. (2014). Freedom through constraints: User-oriented architectural design. *Advanced Engineering Informatics*, *28*(1), 28–36. https://doi.org/10.1016/j.aei.2013.11.003

Niklasch, C., Frodl, S., Libreros, A. & Neher, H. (2020). Entwicklung eines robusten Kopplungssystems für Querschlagöffnungen bei Tübbingtunneln. *Beton- und Stahlbetonbau*, *115*(S1), 72–80. https://doi.org/10.1002/best.202000078

Ninić, J., Koch, C., Vonthron, A., Tizani, W. & König, M. (2020). Integrated parametric multi-level information and numerical modelling of mechanised tunnelling projects. *Advanced Engineering Informatics*, *43*(March 2019), 101011. https://doi.org/10.1016/j.aei.2019.101011

Obergrießer, M. (2016). *Entwicklung von digitalen Werkzeugen und Methoden zur integrierten Planung von Infrastrukturprojekten am Beispiel des Schienen- und Straßenbaus* (Dissertation). Technische Universität München.

Object Management Group. (2019). *OMG Meta Object Facility (MOF) Core Specification* (Techn. Ber.).

Ohori, K., Ledoux, H., Biljecki, F. & Stoter, J. (2015). Modeling a 3D City Model and Its Levels of Detail as a True 4D Model. *ISPRS International Journal of Geo-Information*, *4*(3), 1055–1075. https://doi.org/10.3390/ijgi4031055

Oosterom, P. V. & Schenkelaars, V. (1995). The Developement of an Interactive Multi-Scale GIS. *International Journal of Geographical Information Systems*, *9*(5), 489–507.

Overbeek, J. F. (2006). *Meta Object Facility (MOF): investigation of the state of the art* (Techn. Ber.). Essay, University of Twente. University of Twente.

Owen, J. C. (1991). Algebraic solution for geometry from dimensional constraints. *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, 397–407.

Pan, Y., Braun, A., Borrmann, A. & Brilakis, I. (2021). Void-growing: a novel Scan-to-BIM method for manhattan world buildings from point cloud. *Proceedings of the 2021 European Conference on Computing in Construction*, *2*, 312–321. https://doi.org/10.35490/EC3.2021.162

Petzold, F. (2004). *Computergestützte Bauaufnahme als Grundlage für die Planung im Bestand - Untersuchungen zur digitalen Erfassung und Modellbildung* (Dissertation). Bauhaus-Universität Weimar. https://doi.org/10.25643/bauhaus-universitaet.52

Pfaltz, J. L. & Rosenfeld, A. (1969). Web grammars. *Proceedings of the 1st international joint conference on Artificial intelligence*, 609–619.

Plump, D. (1998). Termination of Graph Rewriting is Undecidable. *Fundamenta Informaticae*, *33*(2), 201–209. https://doi.org/10.3233/FI-1998-33204

Pratt, M. J. (1997). Extension of STEP for the Representation of Parametric and Variational Models. *CAD Systems Development* (S. 237–250). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-60718-9_17

Pratt, M. J. (1998). Extension of the standard ISO10303 (STEP) for the exchange of parametric and variational CAD models. *Ifipwg5.2/5.3*, 1–12.

Pratt, M. J. (2004). Extension of ISO 10303, the STEP standard, for the exchange of procedural shape models. *Proceedings - Shape Modeling International SMI 2004*, *2004*, 317–326. https://doi.org/10.1109/SMI.2004.1314519

Pratt, T. W. (1971). Pair grammars, graph languages and string-to-graph translations. *Journal of Computer and System Sciences*, *5*(6), 560–595. https://doi.org/10.1016/S0022-0000(71)80016-8

Preidel, C. (2020). *Automatisierte Konformitätsprüfung digitaler Bauwerksmodelle hinsichtlich geltender Normen und Richtlinien mit Hilfe einer visuellen Programmiersprache (Automated code compliance checking of digital building models with regard to applicable standards and* (Dissertation). Technische Universität München.

Purchase, H. (2000). Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, *13*(2), 147–162. https://doi.org/10.1016/S0953-5438(00)00032-1

Ranta, M., Mäntylä, M., Umeda, Y. & Tomiyama, T. (1996). Integration of functional and feature-based product modelling - The IMS/GNOSIS experience. *CAD Computer Aided Design*, *28*(5), 371–381. https://doi.org/10.1016/0010-4485(95)00056-9

Reddy, E. J., Sridhar, C. N. V. & Rangadu, V. P. (2015). Knowledge Based Engineering: Notion, Approaches and Future Trends. *American Journal of Intelligent Systems*, *5*(1), 1–17. https://doi.org/10.5923/j.ajis.20150501.01

Rensink, A. (2004). The GROOVE Simulator: A Tool for State Space Generation. *International Workshop on Applications of Graph Transformations with Industrial Relevance* (S. 479–485). https://doi.org/10.1007/978-3-540-25959-6_40

Rensink, A. & Taentzer, G. (2007). AGTIVE 2007 graph transformation tool contest. *International Symposium on Applications of Graph Transformations with Industrial Relevance*, 487–492.

Romberg, R. (2005). *Gebäudemodell-basierte Strukturanalyse im Bauwesen* (Dissertation). Technische Universität München.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformation* (Bd. 1). World Scientific. https://doi.org/10.1142/9789812384720

Ruiz-Montiel, M., Boned, J., Gavilanes, J., Jiménez, E., Mandow, L. & Pérez-De-La-Cruz, J. L. (2013). Design with shape grammars and reinforcement learning. *Advanced Engineering Informatics*, *27*(2), 230–245. https://doi.org/10.1016/j.aei.2012.12.004

Runge, O., Ermel, C. & Taentzer, G. (2012). AGG 2.0 – New Features for Specifying and Analyzing Algebraic Graph Transformations. *International Symposium on Applications of Graph Transformations with Industrial Relevance* (S. 81–88). https://doi.org/10.1007/978-3-642-34176-2_8

Sacks, R., Eastman, C. M. & Lee, G. (2004). Parametric 3D modeling in building construction with examples from precast concrete. *Automation in Construction*, *13*, 291–312. https://doi.org/10.1016/S0926-5805(03)00043-8

Safdar, M., Jauhar, T. A., Kim, Y., Lee, H., Noh, C., Kim, H., Lee, I., Kim, I., Kwon, S. & Han, S. (2020). Feature-based translation of CAD models with macro-parametric approach: issues of feature mapping, persistent naming, and constraint translation. *Journal of Computational Design and Engineering*, *7*(5), 603–614. https://doi.org/10.1093/jcde/qwaa043

Salimzadeh, N., Vahdatikhaki, F. & Hammad, A. (2020). Parametric modeling and surface-specific sensitivity analysis of PV module layout on building skin using BIM. *Energy and Buildings*, *216*, 109953. https://doi.org/10.1016/j.enbuild.2020.109953

Schaefer, J. & Rudolph, S. (2005). Satellite design by design grammars. *Aerospace Science and Technology*, *9*(1), 81–91. https://doi.org/10.1016/j.ast.2004.08.003

Schätzle, J. (2016). Evaluate How the STEP Standard AP 242 Could Enable Knowledge Transfer between CAD and KBE Environments. *Master's Thesis, Norwegian University of Science and Technology.*

Schiffer, S. (1998). *Visuelle Programmierung: Grundlagen und Einsatzmöglichkeiten.* Addison-Wesley.

Schubert, M. (2009). *Mathematik für Informatiker.* Vieweg+Teubner. https://doi.org/10.1007/978-3-8348-9585-1

Schuh, G. (2014). *Produktkomplexität managen: Strategien-Methoden-Tools.* Carl Hanser Verlag GmbH Co KG.

Schultz, C., Bhatt, M. & Borrmann, A. (2015). Bridging qualitative spatial constraints and feature-based parametric modelling: Expressing visibility and movement constraints. *Advanced Engineering Informatics.* https://doi.org/10.1016/j.aei.2015.10.004

Sester, M. (2005). Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science*, *19*(8-9), 871–897. https://doi.org/10.1080/13658810500161179

Sester, M. (2001). Maßstabsabhängige Darstellungen in digitalen räumlichen Datenbeständen. *Vol. 544, Reihe C, Deutsche Geodätische Kommission, Habilitationsschrift, Universiät Stuttgart.*

Shah, J. J. & Mäntylä, M. (1995). *Parametric and feature-based CAD/CAM: Concepts, Techniques and Applications.* ohn Wiley & Sons.

Shah, J. J. & Rogers, M. T. (1993). Assembly modeling as an extension of feature-based design. *Research in Engineering Design*, *5*(3-4), 218–237. https://doi.org/10.1007/BF01608364

Shea, K., Aish, R. & Gourtovaia, M. (2005). Towards integrated performance-driven generative design tools. *Automation in Construction*, *14*(2), 253–264. https://doi.org/10.1016/j.autcon.2004.07.002

Siemens AG. (2022). D-Cubed 2D DCM - 2D Geometric Constraint Solver for high-productivity sketching. https://www.plm.automation.siemens.com/global/de/products/plm-components/2d-dcm.html

Sigalov, K. & König, M. (2017). Recognition of process patterns for BIM-based construction schedules. *Advanced Engineering Informatics*, *33*, 456–472.

Singer, D. (2014). Entwicklung eines Prototyps für den Einsatz von Knowledge-based Engineering in frühen Phasen des Brückenentwurfs. *Master's Thesis, Technische Universität München.*

Slepicka, M. (2019). Umsetzung graphbasierter Methoden zur automatisierten Modellerstellung in parametrischen CAD-Werkzeugen. *Bachelor's Thesis, Technische Universität München.*

Ślusarczyk, G., Grabska, E. & Strug, B. (2017a). A Graph-Based Generative Method for Supporting Bridge Design. *Proc. of the EG-ICE Workshop on Intelligent Computing in Engineering, Nottingham, UK, 2017.*

Ślusarczyk, G., Łachwa, A., Palacz, W., Strug, B., Paszyńska, A. & Grabska, E. (2017b). An extended hierarchical graph-based building model for design and engineering problems. *Automation in Construction*, *74*, 95–102. https://doi.org/10.1016/j.autcon.2016.11.008

Ślusarczyk, G., Strug, B. & Grabska, E. (2021). A Generative Design Method Based on a Graph Transformation System. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (S. 367–378). Springer, Cham. https://doi.org/10.1007/978-3-030-87897-9_33

Spur, G., Ebert, J., Fischer, W., Herter, J., Lehr, U., Materne, J., Pahl, G., Specht, D., Thomas, H. Z., Wietog, J. & Zurlino, F. (1993). *Automatisierung und Wandel der betrieblichen Arbeitswelt.* De Gruyter.

Stachowiak, H. (1973). *Allgemeine Modelltheorie.* Springer Wien.

Steinhage, V. (2001). *Zur automatischen Gebäuderekonstruktion aus Luftbildern* (Dissertation). Universität Bonn.

Stiny, G. (1975). *Pictorial and Formal Aspects of Shape and Shape Grammars.* https://doi.org/10.1007/978-3-0348-6879-2

Stojanovic, V., Trapp, M., Richter, R., Hagedorn, B. & Döllner, J. (2019). Semantic Enrichment of Indoor Point Clouds - An Overview of Progress towards Digital Twinning. *Sousa, JP, Xavier, JP and Castro Henriques, G (eds.), Architecture in the Age of the 4th Industrial Revolution - Proceedings of the 37th eCAADe and 23rd SIGraDi Conference - Volume 2, University of Porto, Porto, Portugal, 11-13 September 2019, pp. 809-818.*

Stroud, I. (2006). *Boundary Representation Modelling Techniques.* Springer London. https://doi.org/10.1007/978-1-84628-616-2

Tamassia, R. (2000). Graph Drawing. *Handbook of Computational Geometry* (S. 937–971). Elsevier. https://doi.org/10.1016/B978-044482537-7/50022-X

Tang, F., Ma, T., Guan, Y. & Zhang, Z. (2020a). Parametric modeling and structure verification of asphalt pavement based on BIM-ABAQUS. *Automation in Construction*, *111*(November 2019), 103066. https://doi.org/10.1016/j.autcon.2019.103066

Tang, L., Ying, S., Li, L., Biljecki, F., Zhu, H., Zhu, Y., Yang, F. & Su, F. (2020b). An application-driven LOD modeling paradigm for 3D building models. *ISPRS Journal of Photogrammetry and Remote Sensing*, *161*(January), 194–207. https://doi.org/10.1016/j.isprsjprs.2020.01.019

Theiler, M. & Smarsly, K. (2018). Parametric Information Modeling of Cyber-Physical Systems based on Industry Foundation Classes. *The 16th International Conference on Computing in Civil and Building Engineering (ICCCBE)*.

Tittmann, P. (2003). *Graphentheorie: Eine anwendungsorientierte Einführung*. Carl Hanser Verlag.

Tolman, F. P. (1999). Product modeling standards for the building and construction industry: Past, present and future. *Automation in construction*, *8*(3), 227–235. https://doi.org/10.1016/S0926-5805(98)00073-9

Tovey, M. (1989). Drawing and CAD in industrial design. *Design Studies*, *10*(1), 24–39. https://doi.org/10.1016/0142-694X(89)90022-7

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

Turner, J. (1955). Maxwell on the method of physical analogy. *British Journal for the Philosophy of Science*, *6*(23), 226–238. https://doi.org/10.1093/bjps/VI.23.226

Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *Journal of the ACM (JACM)*, *23*(1), 31–42. https://doi.org/10.1145/321921.321925

Ullmann, J. R. (2011). Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *ACM Journal of Experimental Algorithmics*, *15*(1). https://doi.org/10.1145/1921701.1921702

Ushakov, D. (2012). Russian National 3D Kernel. http://isicad.net/articles.php?article_num=15189

Vajna, S., Weber, C., Zeman, K., Hehenberger, P., Gerhard, D. & Wartzack, S. (2018). *CAx für Ingenieure*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54624-6

van Amstel, F. M., Hartmann, T., van der Voort, M. C. & Dewulf, G. P. (2016). The social production of design space. *Design Studies*, *46*, 199–225. https://doi.org/10.1016/j.destud.2016.06.002

Verhagen, W. J., Bermell-Garcia, P., van Dijk, R. E. & Curran, R. (2012). A critical review of Knowledge-Based Engineering: An identification of research challenges. *Advanced Engineering Informatics*, *26*(1), 5–15. https://doi.org/10.1016/j.aei.2011.06.004

Verroust, A., Schonek, F. & Roller, D. (1992). Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, *24*(10), 531–540.

Vilgertshofer, S. & Borrmann, A. (2015). Automatic detailing of parametric sketches by graph transformation. *Proc. of the 32nd International Symposium on Automation and Robotics in Construction and Mining*.

Vilgertshofer, S. & Borrmann, A. (2016a). A graph transformation based method for the semi-automatic generation of parametric models of shield tunnels. *23rd International Workshop of the European Group for Intelligent Computing in Engineering, EG-ICE 2016*.

Vilgertshofer, S. & Borrmann, A. (2018). Supporting feature-based parametric modeling by graph rewriting. *Proc. of the 35nd International Symposium on Automation and Robotics in Construction and Mining*.

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

Vilgertshofer, S., Jubierre, J. R. & Borrmann, A. (2016b). IfcTunnel - A proposal for a multi-scale extension of the IFC data model for shield tunnels under consideration of downward compatibility aspects. *Proc. of the European Conference on Product and Process Modeling (ECPPM)*.

Volz, S. (2006). *Modellierung und Nutzung von Relationen zwischen Mehrfachrepräsentationen in Geo-Informationssystemen* (Dissertation). Universität Stuttgart.

Wesley, M. A. & Markowsky, G. (1984). Generation of Solid Models from Two-Dimensional and Three-Dimensional Data. In M. S. Pickett & J. W. Boyse (Hrsg.), *Solid Modeling by Computers* (S. 23–51). Springer US. https://doi.org/10.1007/978-1-4613-2811-7_2

Wieber, M. S. (2015). *Qualitätssicherung von Modelltransformationen - Über das dynamische Testen programmierter Graphersetzungssysteme* (Dissertation). Technische Universität. Darmstadt.

Yabuki, N., Aruga, T. & Furuya, H. (2013). Development and application of a product model for shield tunnels. *Proceedings of the 30th ISARC*.

Yabuki, N., Lebegue, E., Gual, J., Shitani, T. & Zhantao, L. (2006). International collaboration for developing the bridge product model IFC-Bridge. *Proc. of the 11th Int. Conf on Computing in Civil and Building Engineering*.

Yang, J., Han, S., Cho, J., Kim, B. & Lee, H. Y. (2013). An XML-Based Macro Data Representation for a Parametric CAD Model Exchange. *CAD Solutions LLC*, *1*(1-4), 153–162. https://doi.org/10.1080/16864360.2004.10738254

Ying, H. & Lee, S. (2021). Generating second-level space boundaries from large-scale IFC-compliant building information models using multiple geometry representations. *Automation in Construction*, *126*(March), 103659. https://doi.org/10.1016/j.autcon.2021.103659

Zhao, A., Xu, J., Konaković-Luković, M., Hughes, J., Spielberg, A., Rus, D. & Matusik, W. (2020). RoboGrammar: Graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics*, *39*(6). https://doi.org/10.1145/3414685.3417831

Zinsmaier, M., Brandes, U., Deussen, O. & Strobelt, H. (2012). Interactive Level-of-Detail Rendering of Large Graphs. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2486–2495. https://doi.org/10.1109/TVCG.2012.238

Zündorf, A. (1992). *Implementation of the Imperative/Rule Based Language PROGRES* (Techn. Ber.). Aachen, Germany, RWTH Aachen.