Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

BIM-basierte Auswertung von Bestandsbauten mit Hilfe von Visual Programming

Rediet Tadesse

Bachelor's Thesis
für den Bachelor of Science Studiengang Bauingenieurwesen

Autor: Rediet Tadesse

Matrikelnummer: [redacted]

1. Betreuer: Prof. Dr.-Ing. André Borrmann

2. Betreuer: Cornelius Preidel, M.Sc.

Ausgabedatum: 01. Oktober 2015

Abgabedatum: 07. März 2016

Abstract

Since the beginning of programming visual development environments have been already a concept in the computer science world. However, the construction industry has recently recognized the potential of this method. While visual programming in computer graphics is mainly used for modeling of difficult geometries, it brings in the construction sector furthermore the advantage, that non-professionals can make a deep intervention in a BIM-model. At the beginning this work defines the basic concepts about BIM and Visual Programming. After that, it approaches the study of possible uses of the graphical programming language in the construction industry. For this reason practical evaluation scenarios are made on a BIM-model of a building of the Technical University of Munich and assess on the compatibility of the visual programming to civil engineering tasks. The software “Revit 2016” is used for modeling while the Add-In “Dynamo für Revit 1.0 (Pre-Release)” was chosen for the evaluations. This Bachelor’s Thesis finishes with the advantages and disadvantages of this technology, and gives an outlook on future-oriented applications in the building industry.

Zusammenfassung

Visuelle Entwicklungsumgebungen sind bereits seit den Anfängen der Programmierung ein Begriff in der Informatikwelt. Für die Bauindustrie hat man aber erst vor kurzem das Potential dieser Methode erkannt. Während visuelle Programmierung in der Computergrafik hauptsächlich zum Modellieren von schwierigen Geometrien verwendet wird, bringt sie in der Baubranche zusätzlich den Vorteil, dass Laien einen tiefen Eingriff in einem BIM-Modell vornehmen können. Diese Arbeit definiert zu Beginn die grundlegenden Begrifflichkeiten zum Thema BIM und Visual Programming und befasst sich anschließend mit der Untersuchung der Einsatzmöglichkeiten der grafischen Programmiersprache in der Baubranche. Dabei werden am BIM-Modell eines Gebäudes der TU München praxisorientierte Auswertungsszenarien durchgeführt und die Kompatibilität der visuellen Programmiersprache auf bauingenieurstechnische Aufgabenstellungen bewertet. Beim Modellieren wird die Software Revit 2016 von Autodesk und für die Auswertungen das Zusatzmodul „Dynamo für Revit 1.0 (Pre-Release)“ verwendet. Den Abschluss dieser Bachelor’s Thesis bilden die Vor- und Nachteile dieser Technologie, sowie ein Ausblick auf zukunftsorientierte Einsatzmöglichkeiten in der Bauindustrie.

.

Inhaltsverzeichnis

1. Einführung und Motivation

Die Digitalisierung in der Bauindustrie schreitet immer weiter voran. Entwürfe und Berechnungen, die früher noch mit der Hand durchgeführt wurden, werden heutzutage fast ausschließlich am Computer erstellt. Die Arbeit mit Stift und Papier wird in einer zunehmend vernetzten Welt überflüssig und weicht der schnelleren und wirtschaftlicheren computerbasierten Methode. In USA und Großbritannien werden mit Building Information Modelling (BIM) bereits intensiv Gebäudedaten ausgetauscht. Während in diesen Ländern der Datenaustausch mit BIM fest in den Richtlinien und Normen verankert ist, findet der Einsatz von BIM in Deutschland nur vereinzelt statt. Dabei sind die Vorteile von BIM vielseitig (Borrmann, et al., 2015).

Die Möglichkeit, die die Information eines Gebäudes in einem Modell bietet, liegt nicht nur beim Austausch von Daten zwischen Projektbeteiligten, sondern findet ebenfalls mit externen Berechnungs- und Analyseprogrammen statt. Zum intuitiven Programmieren privater Software haben Softwarehersteller seit einigen Jahren Visual Programming Languages (VPLs) entwickelt. Durch den Einsatz visueller Elemente wird mit Hilfe VPLs, Nicht-Programmieren das „Schreiben“ eigenständiger Codes und somit die Entwicklung individueller Programme ermöglicht (Schiffer, 2001).

Im Zuge von BIM wurde seit kurzem das Potential der visuellen Programmierung im Bauwesen erkannt. Die zukunftsorientierte Bedeutung und die verschiedene Anwendungsmöglichkeiten für BIM-Modelle kennzeichnen die große Motivation hinter dieser Arbeit.

1.1 Einleitung

Der Vorteil von BIM ist, dass im Gebäudemodell nicht nur die rein geometrische, sondern auch die semantische Information enthalten ist. Mit Hilfe dieser Informationen ist es beispielsweise möglich, Berechnungen, Analysen und Simulationen durchzuführen. Dafür werden externe Softwares verwendet, die auf die Programmierschnittstelle (engl.: Application Programming Interface; kurz: API) einer Modellierungssoftware zugreifen, um entsprechende Daten abzurufen. Der Benutzer solcher Software ist aufgrund der angebotenen Programme automatisch in seinen Anwendungsmöglichkeiten beschränkt. Bei individuellen Wünschen des Anwenders, um speziellen Auswertungen anzufordern, liefern geschlossene Systeme kaum ideale Lösungen. Deswegen muss der textuelle Code aus der API direkt manipuliert und ausgewertet werden. Die Anwender von BIM-Tools sind überwiegend Architekten und Bauingenieure, die üblicherweise nur in den seltensten Fällen ausreichende Programmierkenntnisse haben. Um aber automatisierte Prozesse zu programmieren, ist der Zugriff auf die Programmierschnittstelle einer Modellierungssoftware notwendig.

Eine aus der Kognitionspsychologie bekannte Erkenntnis findet bereits seit einigen Jahren in der Welt der Informatik Anwendung. Sie besagt, dass visuelle Informationen im Gehirn schneller aufgenommen werden als textuelle. Auch Informatiker schreiben ihre Programme hauptsächlich nur in textueller Form, denn ein Text-Code beschreibt die Arbeits- und Funktionsweise eines Programms sehr genau.

Bei komplizierten Strukturen und Abläufen sind solche Codes auch für Programmierer schwer nachvollziehbar. Mit der kognitionspsychologischen Wissen, die auch auf die Kommunikation zwischen Mensch und Maschine anwendbar ist, entwickelte man Visuelle Programmiersprachen. Wie sich schnell zeigte, brachte die Einführung einer grafischen Programmierumgebung auch Menschen mit weniger Programmierkenntnissen, die Möglichkeit der Erstellung eigener Programme (Schiffer, 2001).

Inzwischen ist mit der Ausbreitung von Building Information Modeling die Entwicklung von VPLs auch im Bauwesen vorangeschritten und gewinnt stetig an Bedeutung.

1.2 Ziel der Arbeit

Ziel dieser Bachelorarbeit ist die Untersuchung der Einsatzmöglichkeiten, der sich noch in der frühen Entwicklung befindenden Visuellen Programmierumgebung in der Baubranche. Am BIM-Modell eines bestehenden Gebäudes werden praxisorientierte Auswertungsszenarien durchgeführt und die Kompatibilität der visuellen Programmiersprache bewertet. Dabei wird das Augenmerk speziell auf die visuelle Umsetzbarkeit von Aufgabenstellungen aus dem Bauwesen gelegt.

1.3 Aufbau der Arbeit

Bevor die Auswertungsszenarien beschrieben werden, behandelt diese Arbeit zunächst die Themen BIM-Modellierung und Visual Programming jeweils im Detail.

Im folgenden Kapitel werden zunächst grundlegende Verfahren der geometrischen Modellierung besprochen. Dabei werden zwei unterschiedliche Repräsentationen zur Beschreibung von Volumenkörpern, sowie die Grundlagen der parametrischen Modellierung gezeigt. Anschließend wird die Beschreibung und die Darstellung der semantischen Information eines BIM-Modells behandelt.

Kapitel 3 geht schließlich auf die Thematik der visuellen Programmierung ein. Hier wird eine genaue Definition für den Begriff „visuell“ bei der visuellen Programmierung festgelegt. Darauf aufbauend werden die Eigenschaften von visuellen Programmierumgebungen vorgestellt.

Im Anschluss werden in Kapitel 4 die zuvor erklärten Konzepte der BIM-Modellierung und des Visual Programming anhand eines Beispielszenarios demonstriert. Dafür wird ein Gebäude der TU München modelliert und auf Grundlage dieses BIM-Modells werden bautypische Auswertungen vorgeführt. Zur Bearbeitung des praktischen Teils, kommen die marktführenden Softwares Revit und Dynamo für Revit der Firma Autodesk zum Einsatz.

Den Schluss dieser Arbeit bildet eine Bewertung der Visuellen Programmierung. Es werden Vor- und Nachteile dieser Innovation speziell für die Baubranche aufgezählt. Schließlich wird die Umsetzung von Dynamo für Revit im Einzelnen kritisch betrachtet.

2. Building Information Modeling

Folgendes Kapitel bezieht die Quellen aus (Borrmann, et al., 2015).

2.1 Begriffsdefinition

Der Begriff Building Information Modeling (kurz: BIM) beschreibt das digitale, dreidimensionale Abbild eines Gebäudes. BIM-Modelle haben den Vorteil, dass sie vordefinierte bauspezifische Objekte, wie Wände, Türen und Fenster enthalten. Verknüpft mit weiteren Informationen können diese Gebäudemodelle unter anderem in Analyse- und Simulationsprogramme eingebaut werden. Diese digitalen Bauwerksmodelle besitzen eine Datenbank rund um das Bauwerk, um in jeder Lebensphase eines Bauwerks Informationen bereitzustellen. Die Wichtigkeit von BIM in der Baubranche wird vor allem bei Großprojekten sichtbar. Viele Planungskatastrophen bei Großprojekten belegen, in einer Bauindustrie mit immer höheren Anforderungen, die Notwendigkeit dieser Art der Vernetzung. Die schwer überschaubaren Aufgabenverteilungen und die Notwendigkeit einer Zusammenarbeit zwischen den Beteiligten erfordert daher ein System, der den Austausch und die Speicherung von Informationen erlaubt.

2.2 Geometrische Modellierung

2.2.1 Geometrische Repräsentationen

Das Grundgerüst eines BIM-Modells ist seine dreidimensionale Geometrieabbildung. Die geometrische Modellierung von Festkörpern erfolgt in zwei unterschiedlichen Varianten: Mit dem expliziten und dem implizitem Verfahren. Beim expliziten Verfahren werden Körper über ihre Oberfläche beschrieben. Implizite Verfahren nutzen hingegen den Konstruktionsprozess eines Modells. In einem Building Information Model werden beide Verfahren zur Darstellung von Geometrien hergenommen.

Explizites Verfahren

Das explizite Verfahren nutzt die Methoden der Boundary Representation (BRep). Dabei werden in einer hierarchischen Struktur die Randelemente von untergeordneten Elementen mit unterschiedlichen Methoden beschrieben, weswegen diese Repräsentation auch Randdarstellungsverfahren genannt wird. Als Basiskomponenten werden dabei Flächen (faces), Kanten (edges) und Vektoren (vertices) verwendet. Die Beschreibung eines Körpers erfolgt nach folgendem Prinzip: Punkte werden über Vektoren, Kanten über Punkte, Flächen über Kanten und Körper über Flächen dargestellt. Diese Verkettungsreihe bildet den topologischen Teil eines Körpers. Abhängig vom Geometrie-Kernel können weitere Einteilungen dieser Beziehungen vorgenommen werden. Für komplizierte Geometrien sind genauere Beschreibungen notwendig. Beispielsweise müssen anstatt gerader Kanten gekrümmte Kurven – sogenannte Splines – abgeleitet und zur Beschreibung von Hohlräumen zusätzlich *Lumps* und *Shells* eingesetzt werden. Abbildung 1 zeigt das Datenmodell des ACIS-Kernels, dass für CAD- und BIM-Software verwendet wird.

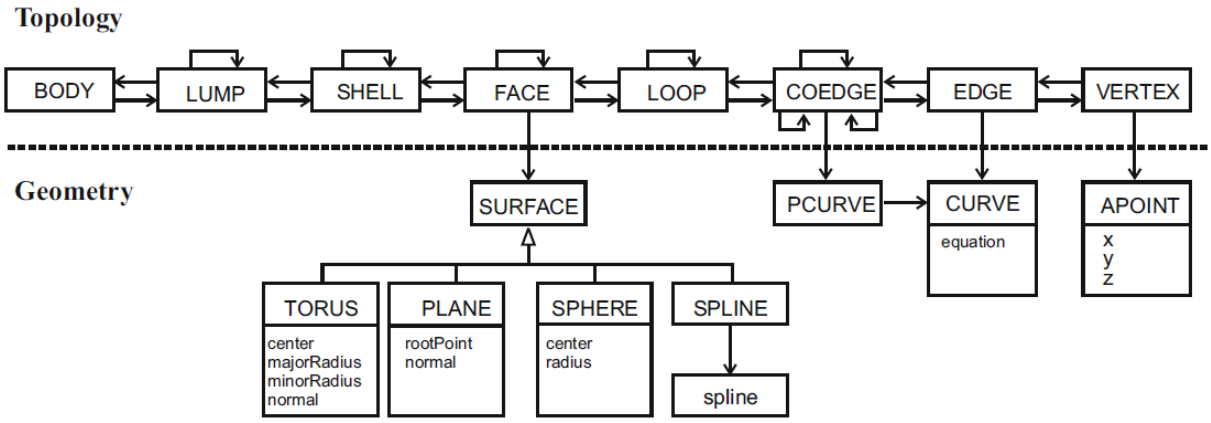

Abbildung 1: Das Datenmodell des ACIS-Geometriekernels (Borrmann, 2015)

Implizites Verfahren

Eine andere Möglichkeit der geometrischen Repräsentation besteht im impliziten Verfahren. Im Gegensatz zum expliziten Verfahren wird hier nicht das Ergebnis eines Modells, sondern sein Entstehungsvorgang beschrieben. Man unterscheidet im Wesentlichen zwischen der Modellierung mit Extrusionen und Rotationen und der Anwendung der Constructive Solid Geometry (CSG) Methode.

Volumenbeschreibung mit der CSG-Methode

CSG verwendet boolesche Operationen zwischen primitiven Körpern. Diese Primitive sind meistens Quader, Kugel, Zylinder und Prismen. Zusätzlich enthalten diese Primitivkörper parametrische Abmessungen, die individuell angepasst werden können. In Abbildung 2 sieht man beispielhaft die typische Entstehungskonstruktion bei einem CSG-Verfahren. Auf der linken Seite befindet sich das fertige Modell. Es besteht aus den Elementen auf der rechten Seite, die mit Hilfe einer Vereinigung zweier Quader und der Differenz eines Zylinders entstanden sind. CSG hat sich zwar als vorteilhaft erwiesen; zur Beschreibung komplexer Geometrien ist diese Methode jedoch nicht flexibel genug. Deshalb erlauben moderne CAD- und BIM-Programme Mengenoperatoren, mit beliebig vom Anwender modellierten dreidimensionalen Formen.

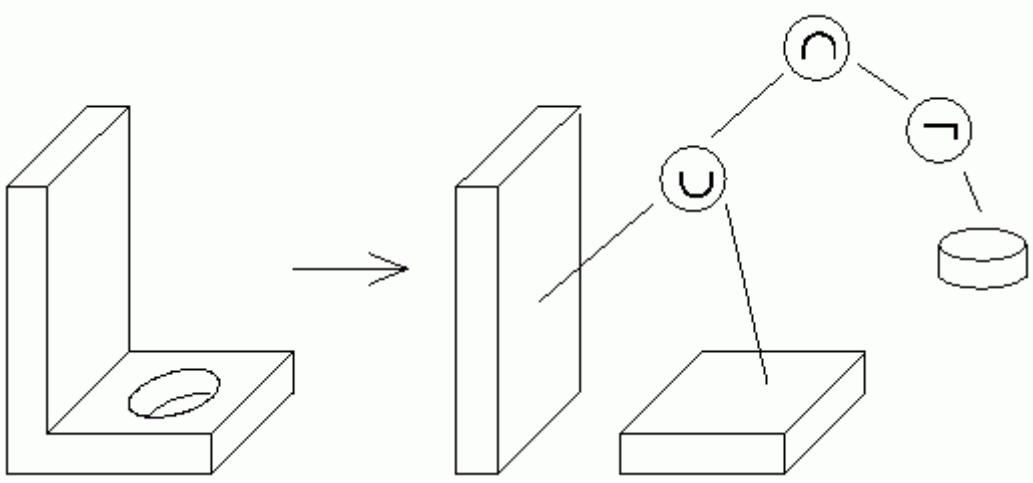

Abbildung 2: fertiges Modell und dessen CSG-Aufbau (Berlin, kein Datum)

Volumenbeschreibung mit Extrusion und Rotation

Im Gegensatz zur CSG-Methode mit dreidimensionalen Primitivkörper, wird in vielen CAD- und BIM-Systemen die 3D-Darstellung aus einer 2D-Form erschlossen. Dazu wird eine Fläche oder ein geschlossenes Polygon nach einem vorgegebenen Weg im dreidimensionalen Raum geführt und aus dem eingenommenen Raum das 3D-Modell erzeugt. Diese Form der Geometriepräsentation nennt sich Extrusion und Rotation. Man unterscheidet dabei zwischen vier Arten:

Extrusion: Führung der 2D-Form entlang einer geraden Kurve

Rotation: Rotieren der 2D-Form um eine Achse

Sweep: Führung der 2D-Form entlang einer gekrümmten Kurve

Lofting: (auch Sweep-Verschmelzung) Führung einer 2D-Form entlang einer Kurve, während sich die 2D-Form linear zu anderen vorgegebenen 2D-Formen auf dem Pfad transformieren

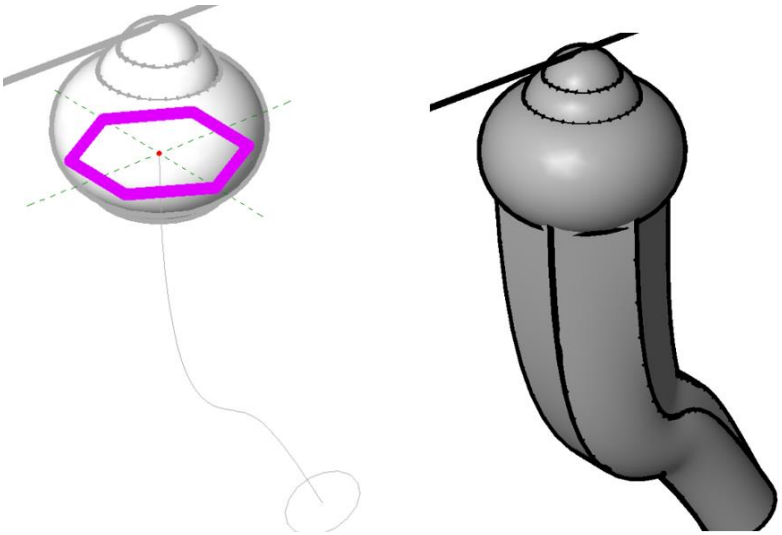

Abbildung 3: Lofting – Transformation eines 6-Ecks zu einem Kreis

2.2.2 Parametrische Modellierung

Bei parametrischen Modellen befinden sich Teile eines Modells so in Abhängigkeiten und Zwangsbedingungen zueinander, dass durch Ändern von Parameterwerten aufwandsarme und schnelle Anpassungen vorgenommen werden können (Borrmann). Der Anwender kann die Abhängigkeiten aus den geometrischen Eigenschaften des Modells oder über Beziehungen und Formeln beschreiben (Abbildung 4). Diese Form der parametrischen Modellierung wird in BIM-Systemen bei der Modellierung und Platzierung von bauspezifischen Objekten verwendet. Für einen simplen Zugriff auf wesentliche Eigenschaften eines BIM-Modells stellen Parameter die wichtigsten Instrumente dar. Man unterscheidet zwischen zwei Arten von Zwangsbedingungen: geometrische Constraints, die Beziehungen zu anderen Elementen (wie Parallelität, Lot, Verknüpfung, usw.) definieren und Abmessungs-Constraints, die ausschließlich geometrische Größen (Länge, Winkel, usw.) parametrisieren. In Abbildung 5 sind in einem BIM-Modell diese beiden Fälle dargestellt. Die blauen Schlosssymbole zeigen geometrische Zwangsbedingungen, die mit Referenzebenen und Rändern verknüpft werden können, während die Bemaßungen Abmessungs-Constraints besitzen.

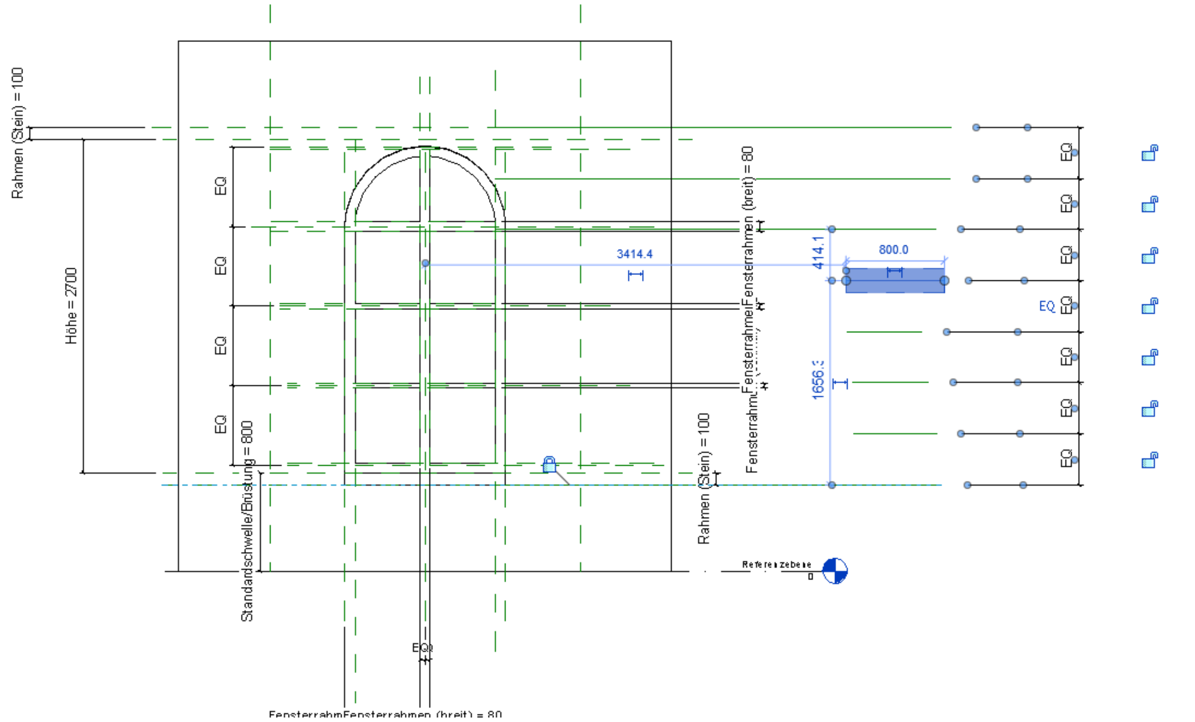

Abbildung 5: Parametrisiertes BIM-Fenster

Parameter	Wert	Formel	Sperren
Konstruktion			
Wandabschluss	Nach Basisbauteil	=	
Konstruktionstyp		=	
Abmessungen			
Fensterrahmen (breit)	80.0	=	☐
Fensterrahmen (schmal)	40.0	=	☐
Höhe	2700.0	=	☑
Breite	1300.0	= Höhe * 13 mm / 27 mm	☑
Rahmen (Stein)	250.0	=	☐
Rohbaubreite	2700.0	= Höhe	☑
Rohbauhöhe	1300.0	= Breite	☑
Analytische Eigenschaften			
Analytische Konstruktion	<Keine Auswahl>	=	
Durchlässigkeit für sichtbares Licht		=	
Gesamtenergiedurchlassgrad		=	
Thermischer Widerstand (R)		=	
Wärmeübergangskoeffizient (U)		=	
IFC-Parameter			

Abbildung 4: Parametertabelle einer BIM-Fensterfamilie

Parametrisches Modellieren wird hauptsächlich in reinen 3D-Modellen in vollem Umfang angeboten. Die Funktionen, die derzeit für BIM-Systeme angeboten werden, reichen dennoch für flexible Anpassungen des Modells aus.

2.3 Objektorientierte Modellierung

Objekte in Building Information Models enthalten zusätzlich zu ihrer Geometrie noch semantische Informationen. Die Geometrie definiert das äußere Erscheinungsbild des Modells, wohingegen man unter der Semantik eines Gebäudemodells die Beziehung von Objekten zu anderen Objekten, wie beispielsweise zu Materialien, Bauteilen oder externe Einwirkungen versteht. Zur digitalen Beschreibung und Strukturierung dieser realen Informationen hat sich die Methode des objektorientierten Modellierens (OOM) etabliert. Sie wird verwendet, um komplexe Zusammenhänge durch gezielte Auswahl signifikanter Elemente zu abstrahieren.

Ein digitales Bauwerk besteht aus Objekten, die wiederum aus untergeordneten Objekten zusammengesetzt sind. Diese Objekte werden durch statische, strukturelle Merkmale (Attribute) und dynamische, verhaltensbezogene Merkmale (Methoden) beschrieben. Für den Datenaustausch werden nur die Attribute eines Modellobjekts herangezogen.

2.3.1 OOM-Methodik

Das objektorientierte Modellieren durchläuft drei Phasen:

OOA-Phase: Klärung des zu untersuchenden Objekts und des zu erwartenden Softwarefunktionalitäten

OOD-Phase: Untersuchung des Modells nach existenten Objekten und Beziehungsverhältnissen zu anderen Objekten

OOP-Phase: Implementierung des Modells

Die Unified Modeling Language (UML) ermöglicht als grafische Modellierungssprache die Beschreibung dieses objektorientierten Modells mittels verschiedenen Diagrammarten.

2.3.2 Objektorientierte Basiskonzepte

Objekte und Klassen

Während ein Objekt die Abbildung eines Gegenstands aus der Realität verkörpert, ist eine Klasse das Typ eines Objekts, die das Verhalten und die Struktur gleichartiger Objekte beschreibt. Als Instanz einer Klasse wird ein Objekt durch seinen Zustand, sein Verhalten und seiner Identität gekennzeichnet.

Attribute und Methoden

Die statische Beschreibung eines Objektzustands findet mit Hilfe von Attributen statt. Sie beschreiben die Eigenschaften eines Objekts und seine Objektstruktur. Man unterscheidet zwischen Attributen, die alle Objekte einer Klasse auf gleicher Weise besitzen und ihren individuellen Attributwerten. Attribute werden mit einem Namen und einem Datentyp gekennzeichnet. In Abbildung 6 besitzt „Wand Nr. 17“ als Attributnamen „ID“ und als Attributwert eine individuelle Identifikationsnummer. Eine andere Wand der gleichen Klasse hat zwar ebenfalls eine ID, aber nicht dieselbe ID-Nummer. Methoden dagegen dienen dazu, das Verhalten eines Objekts zu definieren. Da sie den ausführbaren Programmcode besitzen und von Klassen mit Attributen zusammengefasst werden, erfolgt der Lese- und Schreibzugriff auf Attribute indirekt über Methoden.

Vererbung

Neben den Attributen spielen die Beziehungen zwischen Objekten eine entscheidende Rolle für die Semantik eines Objektmodells. Ein wesentliches Konzept zur Modellierung von Beziehungen zwischen Objekten ist die Vererbung. Vererbung bedeutet, dass eine spezialisierte Unterklasse Eigenschaften (Attribute) einer oder mehrerer allgemeiner Oberklassen erbt. Besitzen mehrere Objekte im Hinblick auf Beziehungen die gleichen Eigenschaften, kann diese Information anstatt in Form eines Attributs, in Form von Verbungen angeordnet werden. Eigenschaften wie beispielsweise eine Wandöffnung besitzen dann eine Semantik und stellen

eine Oberklasse von Türen und Fenstern dar, während gleichzeitig neue Unterklassen für beide Bauteile erstellt werden.

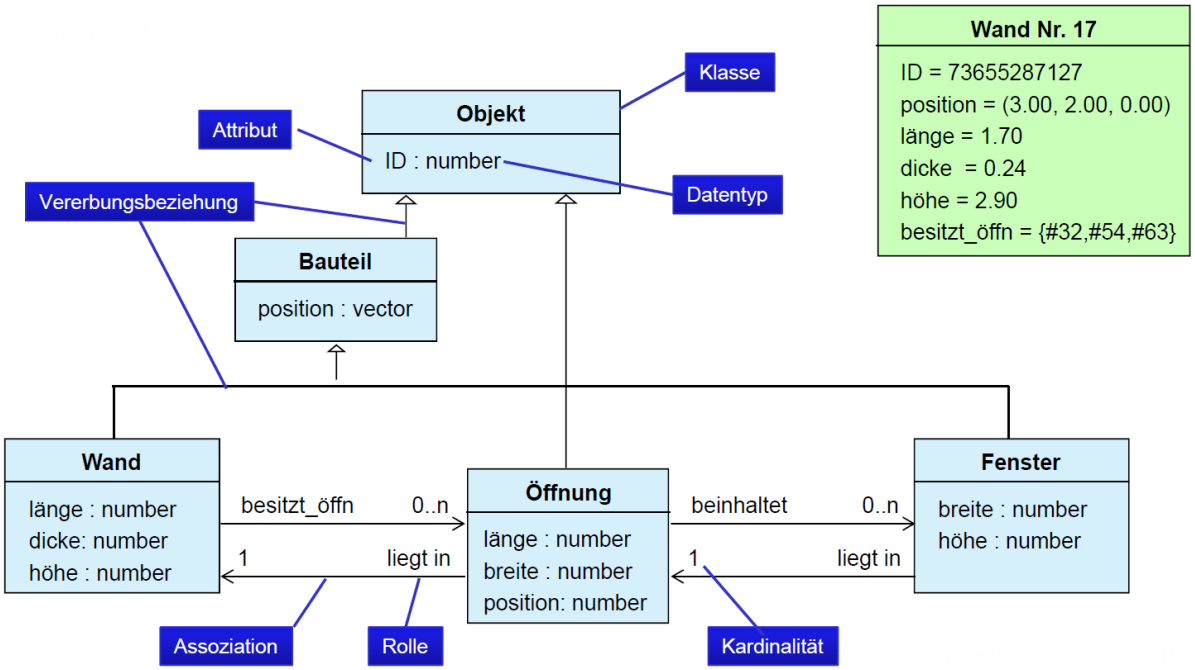

Abbildung 6: Prinzip der UML-Darstellung (Borrmann, 2015)

Assoziationen

Im Gegensatz zur Vererbung liegt bei der Assoziation keine hierarchische Struktur, sondern eine gleichrangige Beziehung zwischen Objekten vor. Auf diese Weise können Daten und Abhängigkeiten unter den Objekten ausgetauscht werden. Zur quantitativen Klärung des Informationsaustauschs werden dabei Kardinalitäten verwendet, die zeigen mit wie vielen Objekten das andere Objekt in Relation steht und umgekehrt. Diese werden mit „1“ oder bei mehrfachen mit „n“ bzw. „0..n“ gekennzeichnet. Auf Assoziationen kann entweder direkt oder über eine sogenannte Assoziationsklasse verwiesen werden. Der Vorteil einer Assoziationsklasse ist die zusätzliche semantische Information.

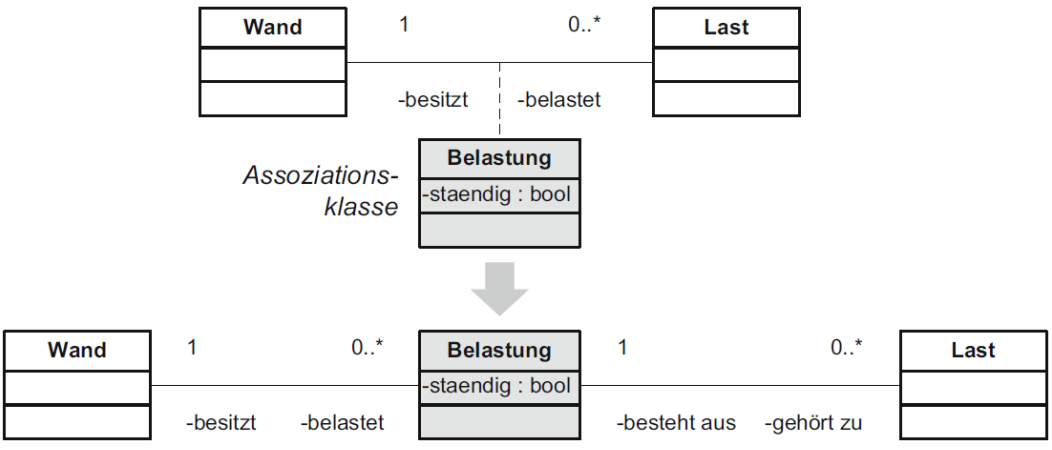

Abbildung 7: Ergänzung um eine Assoziationsklasse einer Assoziation im Beispielmodell (Borrmann, et al., 2015)

3 Visual Programming

Die Kognitionspsychologie beschäftigt sich mit der Analyse der menschlichen Informationsverarbeitung. In diesem Fachbereich stellte Allan Paivio 1986 die duale Kodierungstheorie vor, wonach Menschen Wort- und Objektinformationen auf unterschiedliche Weisen aufnehmen. Zum einen in verbalen und zum anderen in visuellen Kodierungssystemen (Anon., 2016). Die textuelle Informationsaufnahme erfolge im Gehirn stückweise (sequentiell-analytische Verarbeitung), wogegen die Informationsaufnahme von Bildern vom Gehirn parallel verarbeitet wird (Faas, et al., 2010). Nach Auswertung verschiedener Experimente zeigte sich, dass Bilder deutlich schneller verarbeitet werden als Texte und visuelle Informationen somit länger im Gedächtnis bleiben (Schiffer, 2001).

3.1 Definition: Visuell

Seit den Anfängen des VPL wird nach dem gesucht, was die visuelle Programmierung als solches ausmacht und was unter dem Begriffe zu verstehen ist. Eine eindeutige Definition gibt es jedoch noch immer nicht. Fast alles was bei der Mensch-Maschine-Kommunikation mit grafischer Darstellung zu tun hat und grafische Elemente verwendet kann unter der Kategorie VPL eingeordnet werden. Trotzdem kann man zwei große Unterscheidungen bei der Art der Programmiersprache beobachten. Objektorientierte Entwicklungsumgebung mit einem grafischen Editor zur Gestaltung der Benutzerschnittstelle, deren Programmlogik aber textuell erstellt wird (z.B. Visual Basic) und Entwicklungsumgebungen mit visueller Programmiersprache, in der Text nur eine geringe Rolle spielt und vor allem für Beschriftungen, Kommentare sowie Zahlen- und Zeichenkonstanten verwendet wird. Vielen Experten reicht die grafische Darstellung der Benutzeroberfläche nicht aus, um solche Programme visuell zu nennen (Schiffer, 2001).

Zur Beantwortung der Frage, wann ein Programm „visuell“ genannt werden kann, liefert Schiffer eine Definition: Damit ein Handlungsziel erreicht wird, muss es unverzichtbar sein, dass ein Programm auf visuelle Weise mit dem Anwender kommuniziert. Zur Klarstellung dieser Definition stellt er folgende Idee vor:

Eine Programmiersprache stellt Schlüsselwörter fettgedruckt dar. Das einzige Unterscheidungsmerkmal dieser Schlüsselwörter zu anderen Bezeichnern ist nur die Fettschrift. In dieser Sprache würde nun die Anweisung stehen in Pseudocode 1 stehen: Pseudocode 2):

```
if while then do else end
```

Pseudocode 1: Beispiel 1 zur Definition von „visuell“

Die *if-else* – Verzweigung ruft den booleschen Ausdruck *while* aus und führt bei einer wahren Aussage *do* und bei einer falschen *end* aus.

Eine Umstellung der Schlüsselwörter ist in Pseudocode 2 zu sehen:

```
if while then do else end
```

Pseudocode 2: Beispiel 2 zur Definition von „visuell“

Hier wird zunächst die Prozedur *if* aufgerufen. Die *while*-Schleife wertet dann den booleschen Ausdruck *then* aus und ruft die Prozedur *else* so lange auf, bis *then* den Wert falsch ergibt.

Ohne die zusätzliche visuelle Komponente kann das Handlungsziel nicht erreicht werden. Der Computer könnte die Anweisung also nicht ausführen. Mit diesem Beispiel erklärt Schiffer, wie seiner Meinung nach der Begriff „visuell“, im Zusammenhang mit Visual Programming“ zu verwenden ist.

3.2 Konzept der visuellen Programmierung

Visuelle Programmierwerkzuge verwenden eine 2D-Arbeitsebene. Aus einer Bibliothek können Funktionen in Form von Blöcken (engl.: Nodes) ausgewählt und in diese hinzugefügt werden. Ähnlich wie in der Graphentheorie werden diese Blöcke durch einfach gerichtete Kanten miteinander verknüpft. Die Kanten stellen den Weg des Datenflusses zwischen der Ausgabe- und der Eingabevariable von zwei Blöcken dar. Diese Darstellungsart erleichtert Nicht-Programmierern Beziehungen und Funktionen ohne Programmierkenntnisse zu verstehen. Visuelle Programmierumgebungen werden nicht nur als eigenständige Software angeboten, sondern oft auch als Zusatz für andere Programme herausgegeben. Zwischen dem Hauptprogramm und dem Zusatzmodul können Informationen ausgetauscht werden mit denen sich tiefgreifende Veränderungen vornehmen lassen können.

3.1.1 Informationsfluss

Ein Block wird üblicherweise mit einem Namen oder einem Symbol definiert, der die Funktion oder den Typ beschreibt. Er besitzt Eingabevariablen auf einer Seite und die Ausgabevariable auf der anderen. Die Eingabe- und Ausgabevariablen haben meistens Namen, die aussagen, welche Datentypen diese erwarten oder ausgeben. In Abbildung 8 ist in der Mitte der typische Aufbau eines solchen Blocks zu sehen. Hierbei handelt es sich um einen Minus-Operator-Block. Die Variablen A und B sind Eingabevariablen. Variable A erwartet einen Minuenden und B den Subtrahenden, während R als Ausgabevariable das Ergebnis der Subtraktion ausgibt. Die Informationen werden über Verbinder an die entsprechenden Variablen übergeben. Der Informationsfluss erfolgt bei nahezu allen visuellen Programmierwerkzeugen von links nach rechts, weshalb auf eine gerichtete Kante verzichtet wird. In diesem Bild wird das Feld (engl. Array) mit den Zahlen 6, 7, 8 und 12 an A übergeben. Jedes Element in diesem Feld wird dann mit 5 subtrahiert. Die Ausgabe ergibt wieder ein Feld.

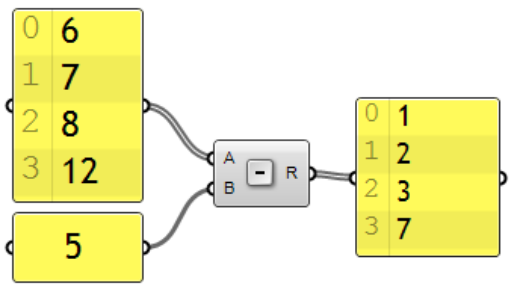

Abbildung 8: Informationsfluss in Grasshopper (Anon., 2016)

In Abbildung 9 und Pseudocode 3 sieht man als Vergleich die visuelle und textuelle Form eines kleinen Programms.

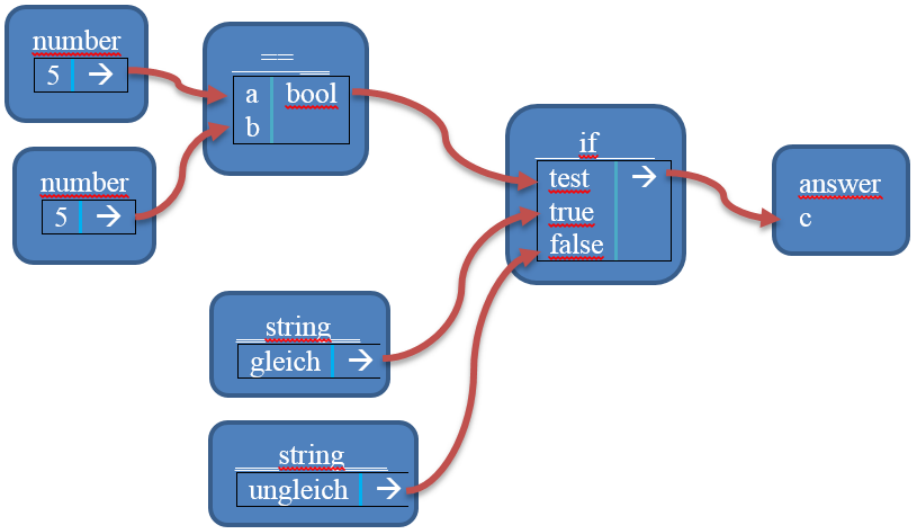

Abbildung 9: Beispiel eines visuellen Programms

```
c = test(5,5);

function c = test (a,b)
      if (a == b)
            c = 'gleich';
      else
            c = 'ungleich';
      end
end
```

Pseudocode 3: Beispiel eines textuellen Programms

Dem == – Block werden zwei Double-Werte übergeben. Es wird überprüft, ob sie gleich sind oder nicht. Das boolesche Ergebnis des Blocks wird an den *if* – Block weitergeleitet. Der *if* – Block gibt, für den Fall, dass die Variable den Wert *true* übergeben bekommt, den String „gleich“ oder bei *false* den String „ungleich“ aus. Es ist nicht möglich den Ausgabewert mit der Eingabevariable eines selben Blocks zu verbinden.

3.1.2 Granularität

Eine weitere wichtige Eigenschaft der visuellen Programmierumgebung ist die Granularität, also die Feinheit von bereitgestellten Blöcken (Ritter, et al., 2015). Es stellt sich die Frage, ob Blöcke mehrere oder nur wenige Teilschritte zusammenfassen. Im oben gezeigten visuellen Code werden beispielsweise nur einfache Grundfunktionalitäten verwendet. Diese Blöcke können kaum noch in weitere Blöcke unterteilt werden. Der Vorteil einer feinen Granularität ist der, dass Programmierer beim Kombinieren von Blöcken weniger Einschränkungen haben, da sie elementare Funktionen verwenden. Gleichzeitig wird durch die Menge der einzelnen Blöcke, das Programm unübersichtlich. Im Gegensatz dazu würde eine visuelle Umgebung mit „groben“ Blöcken – also Blöcke die mehrere Teilschritte enthalten – eine bessere Handhabbarkeit ermöglichen.

Die Funktion *test* kann als eigenständiger Block zusammengefasst werden. Dieser *test* – Block enthielte den Code in der die beiden Zahlen auf Gleichheit überprüft und an die if-Bedingung weitergegeben werden. Durch Verkürzung der Schritte kann der Programmierer den Code so leicht interpretieren. Einige Softwarehersteller bieten die Funktion für die Gruppierung von

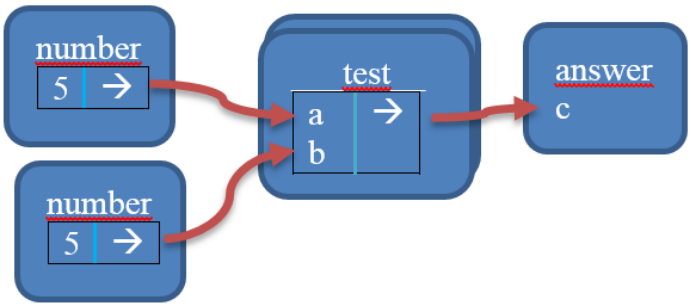

Abbildung 10: Gruppierung mehrerer Blöcke

Blöcken an. Auf diese Weise geben sie dem Benutzer die Möglichkeit den Grad der Granularität selbstständig zu bestimmen. Der zusätzliche Vorteil ist der, dass wiederkehrende Funktionen mit den individuell erstellten Blöcken wiederholt werden können.

3.3 Aktueller Stand

Mit einem ähnlichen Aufbau wie die in der Informatikwelt weit verbreitete VPL LabVIEW brachte Robert McNeel & Associates 2007 Grasshopper auf den Markt. Grasshopper ist inzwischen eine weit verbreitete visuelle Programmierumgebung zum Erstellen von generativen Designs und parametrischen Entwurfsmodellen. Es verwendet Blöcke (auch Knoten genannt) und Kanten zum Verknüpfen von Parametern und geometrischen Größen aus 3D-Modellen, um sie in parametrischen Beziehungen zu setzen (Karen M. Kensek & Douglas Noble, 2014). Die Anwendung ist in der Computergrafik- und CAD-Software Rhinoceros 3D eingebunden und ist nicht kompatibel mit einer anderen Software. Lange Zeit war Grasshopper trotzdem das wichtigste Modellierwerkzeug für Architekten und Designer. Die visuelle Programmierumgebung verlangt keine Programmierkenntnisse und trotzdem ist die Erstellung komplexer Geometrien möglich. Jedoch wird Grasshopper nur während der Entwurfsphase verwendet, da Rhinoceros 3D zwar geometrisch sehr mächtig ist, aber Building Information Modeling nicht unterstützt (Secerbegovic, 2015).

Der Aufwand die rein geometrischen Modelle aus Grasshopper zu exportieren und anschließend in BIM-fähige Modelle umzuwandeln, veranlasste weitere Softwarehersteller visuelle Programmiersprachen speziell für das Bauwesen zu entwickeln. Es sollte möglich sein, sowohl auf die Geometrie als auch auf die Semantik eines Modells zuzugreifen. Mit der Entwicklung von Dynamo hat Autodesk nach der Idee von Grasshopper eine vielversprechende Software entwickelt. Mit dieser können ebenfalls fertige Code-Blöcke so miteinander verknüpft werden, dass sie einen logischen Datenfluss ergeben und Aufgaben wie mathematische Berechnungen und Erstellung komplexer Geometrien ausführen. Autodesk 3dsMax und Maya, Maxon Cinema 4D oder Blender von blender.org sind weitere auf VPL-basierte Softwares, die in der Bauindustrie Einsatz finden (Ritter, et al., 2015).

4 Szenario

Dieses Kapitel behandelt die praktische Anwendung der in Kapitel 2 und 3 gezeigten Konzepte.

4.1 Modellierung in Revit

Viele Hersteller bieten bereits Software für die Modellierung mit Building Information Modeling an. Zu den wichtigsten unter ihnen zählen Revit von Autodesk, Allplan von Nemetschek und Archicad von Graphisoft. Die umfangreichste unter ihnen ist jedoch Revit. Mit der Version 2016 wurde in dieser Bachelorarbeit ein Gebäude der Technischen Universität München als BIM-Modell erstellt.

Revit unterscheidet beim Modellieren grundsätzlich zwischen Projekten und Familien. Ein Projekt stellt ein gesamtes Abbild eines Gebäudes dar. Familien dagegen sind einzelne Elemente (meistens Bauteile), die in Projekte geladen werden können. Diese können nachträglich und global verändert werden.

Bei der Modellierung wurden die in Kapitel 2.2 beschriebenen Methoden angewandt. Dazu gehören die parametrische Modellierung (wie in Abbildung 5 zu sehen) und die CSG-Methoden (wie in Abbildung 11).

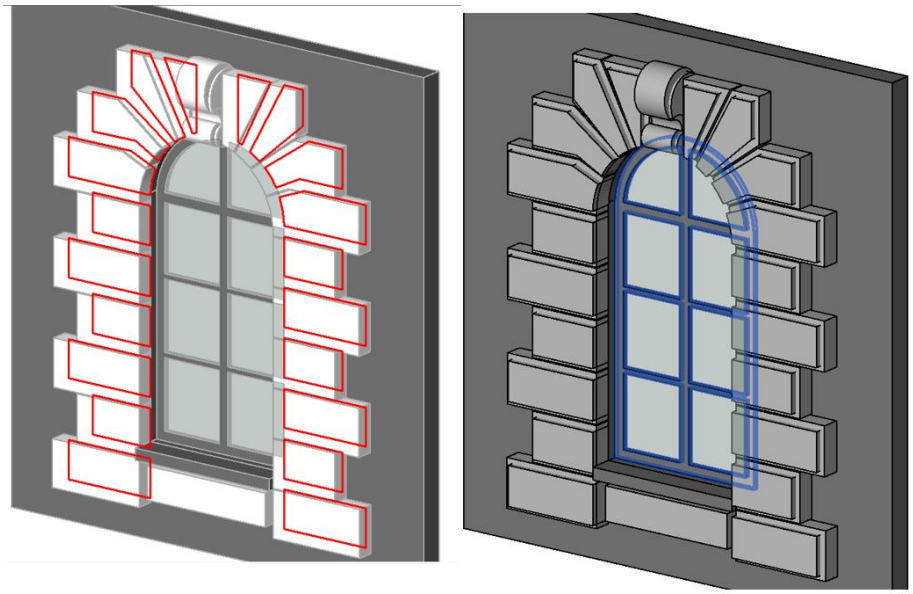

Abbildung 11: Anwendung einer Extrusion in Revit

Als Grundlage wurde ein CAD-Grundriss der Stockwerke 1 bis 3 (.dwg-Datei) verwendet, das in das Modell eingebettet wurde. Höhen wurden mit einem Lasermessgerät bestimmt.

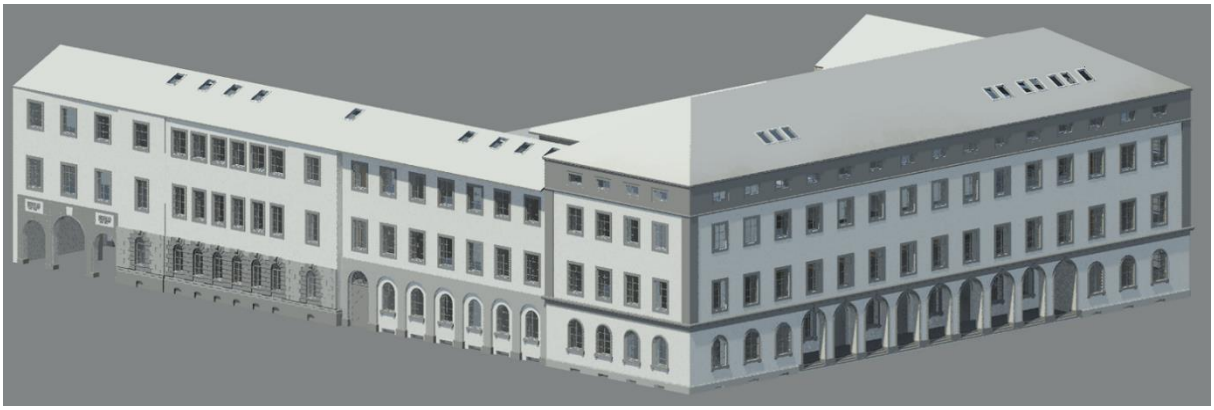

Abbildung 12: Rendering eines Gebäudes der TU München mit Revit

4.2 Dynamo für Revit

Bei Dynamo für Revit handelt es sich um ein quelloffenes visuelles Programmierwerkzeug von Autodesk. Die Software ist speziell für BIM-Modelle entwickelt worden, um dem Benutzer visuelles Programmieren zu ermöglichen. Die stabilste Softwareversion befindet sich aktuell noch in der Beta-Phase (Version 0.9.1) und ist nur als Zusatzmodul für Revit erhältlich. Gleichzeitig bietet aber Autodesk die Software Dynamo Studio an. Diese arbeitet eigenständig und benötigt kein Hauptprogramm. Der Nachteil ist jedoch, dass Dynamo Studio nicht mit Revit kommunizieren kann, weswegen der Funktionsumfang stark eingegrenzt ist.

Mit Hilfe von Dynamo für Revit kann auf Bereiche der Revit-Programmierschnittstelle zugegriffen werden, die über die normalen Revit-Funktionen nicht ansteuerbar sind. Die Software entnimmt uns verarbeitet die Informationen aus dem BIM-Modell. Sie kann dann Änderungen im Hauptprogramm vornehmen. Üblicherweise wird Dynamo für die Erstellung komplexer Geometrien und für Automatisierungen verwendet. Man kann mit dieser Software aber auch Auswertungen und Exporte und Importe in unterschiedliche Formate tätigen. Im Hintergrund arbeitet Dynamo für Revit mit der Programmiersprache Python. Sie ist einer der am weitesten verbreiteten Sprachen, weil sie einerseits einfach zu erlernen, andererseits leicht zu lesen ist. (Wiki) Neben der visuellen Programmierung gibt es in Dynamo auch die Funktion Python-Skripte in textueller Form zu erstellen. Die in Dynamo erstellten Programme werden im .dyn- und .dyf-Format gespeichert.

In dieser Bachelorarbeit zur Vorstellung der visuellen Programmierung Dynamo für Revit in der Pre-Release-Version 1.0.0.20160126T0737 verwendet. Zuerst wird das Arbeiten mit Dynamo erklärt und anschließend BIM-basierte Auswertungen durchgeführt.

4.2.1 Aufbau von Dynamo

Das Fenster von Dynamo ist in vier Bereiche aufgeteilt: Eine Menüleiste im oberen Bereich, eine Bibliothek auf der linken Seite, einem Arbeitsbereich auf der rechten Seite und einer unteren Bar für Ausführungen.

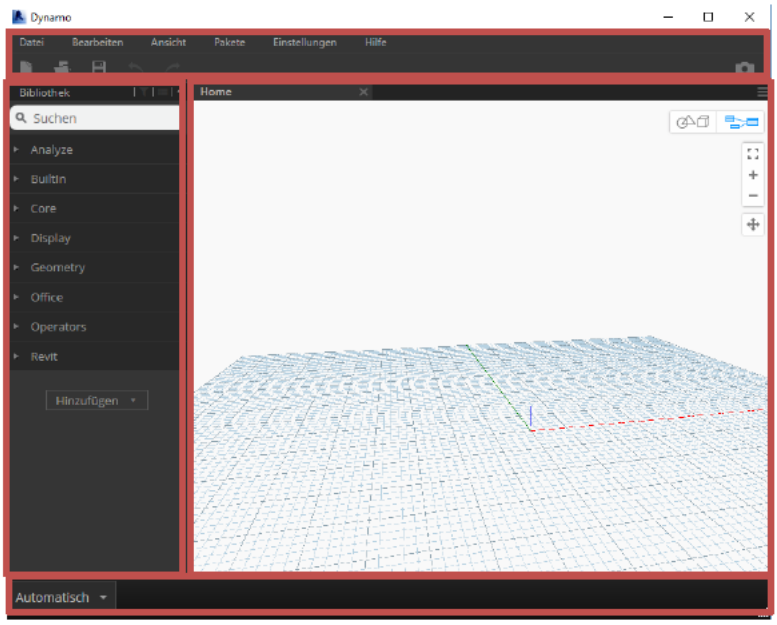

Abbildung 13: Dynamo - Arbeitsfenster

Menü

Die Tabs enthalten neben typischen Funktionen, wie Neu, Speichern, Kopieren und Importieren, zusätzlich die Möglichkeit das Arbeitsfeld als Bild oder das Modell als STL zu exportieren. Unter Bearbeiten kann man Blöcke übersichtlich in Gruppen und Blockgruppen zusammenzufassen. „Ansicht“ enthält Navigations- und Darstellungsfunktionen. In diesem Tab wird im Hintergrund des Arbeitsfeldes eine 3D-Vorschau oder die Revit-Vorschau ein- und ausblendet. Im „Pakete“-Tab können unterschiedliche Pakete heruntergeladen werden. Pakete enthalten eine Reihe von Blöcken zu einem Thema, die innerhalb einer Community erstellt und ausgetauscht werden. Für Anfänger befinden sich im „Hilfe“-Tab kurze Beispiele, die Schritt für Schritt die Bedienung mit Blöcken und einfache und grundlegende Programme zeigen.

Bibliothek

Die Bibliothek besitzt eine Suchleiste mit der nach allen Blöcken gesucht werden kann. Bei der Suche werden nicht nur die Namen der Blöcke sondern auch Beschreibungen durchsucht. Ist der genaue Name und der Aufenthaltsort des Blocks nicht bekannt, kann man diese oft durch Eingeben englischer Schlüsselwörter finden. In der Dynamo Grundeinstellung sind Blöcke in acht Kategorien und Unterkategorien unterteilt. Durch Herunterladen von Paketen oder Erstellen eigener Kategorien kann man diese Anzahl beliebig erweitern. Die wichtigsten Kategorien sind *Core*, *Geometry* und *Revit*. Je nach Funktion sind Blöcke in den Unterkategorien in drei Eigenschaften unterteilt:

Erstellen Aktion Abfrage

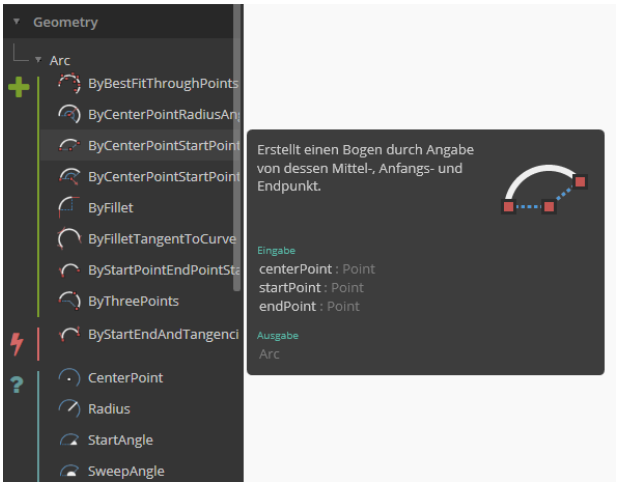

Abbildung 14: Ausschnitt der Dynamo-Bibliothek

Beim Mouse-Over eines Elements wird als Hilfestellung die Funktionsweise des Blocks genauer beschrieben. Zusätzlich werden die Inputs und Outputs des Blocks inklusive der Datentypen der Ein- und Ausgaben aufgelistet.

Arbeitsbereich

Der Arbeitsbereich ist zum einen eine zweidimensionale Ebene zum Erstellen des visuellen Codes (Diagrammansicht) und zum anderen eine 3D-Umgebung, die die in Dynamo erzeugten Elemente in einem Koordinatensystem darstellt (Hintergrund-3D-Vorschau). Das Koordinatensystem und die Einheiten entsprechen denen in Revit. Mit den Navigationsfunktionen in der oberen rechten Ecke kann zwischen den beiden Ansichten navigiert werden. Blöcke, die über die Bibliothek ausgewählt wurden, landen dann in der Diagrammansicht. Die Navigation mit der mittleren Maustaste zum Schwenken und dem Scrollen zum Zoomen erfolgt in beiden Ansichten gleich. In der Diagrammansicht besteht die Möglichkeit mit der linken Maustaste Blöcke auszuwählen, mit der rechten ein Kontextmenü zu öffnen und mit einem Doppelklick einen *Code Block* zu erstellen. Im Kontextmenü befindet sich, für ein schnelleres Arbeiten, die aus der Bibliothek bekannte Suchfunktion. Mit gedrückter rechter Maustaste kann man in der Hintergrund-3D-Vorschau den Raum drehen.

4.2.2 Arbeiten mit Blöcken

Blöcke sind die visuellen Elemente des Programms. In Dynamo besitzen sie je nach Funktion Eingabevariablen auf der linken und Ausgabewerte auf der rechten Seite. Mit einem Doppelklick auf den Namen des Blocks kann man diesen umbenennen. Es gibt unterschiedliche Blockarten: Blöcke, die Werte verarbeiten und dann ausgeben. Welche, mit denen man mit Hilfe eines Textfeldes, eines Buttons, einer Drop-Down-Liste oder eines Schiebereglers eine Auswahl treffen kann. Und Blöcke, die nur Werte annehmen, aber keine ausgeben oder welche, die Situationen und Zustände beschreiben und nur daher keine Eingaben benötigen.

Über die Suchleiste oder den einzelnen Kategorien in der Bibliothek können gewünschte Blöcke in den Arbeitsbereich einfügt werden. Beim Mouse-Over auf eine Variable im Block sieht man meistens eine Erklärung und den zu erwartenden oder ausgegebenen Typ. Zusätzlich

kann die Ausgabe eines Blocks unter jedem Block sichtbar gemacht werden. Durch den Einsatz von Verbindern können Ausgaben mit den Variablen anderer Blöcke verbunden werden.

In Abbildung 15 wird ein Kreis im Raum erzeugt. Der Kreis aus der *Geometry* – Kategorie ist definiert durch einen Radius einer Normalen und einem Mittelpunkt. Der Radius muss einen Double-Wert besitzen und ist folglich mit einem *Number* – Block verbunden, der dem Programmierer die Eingabe rationaler Zahlen erlaubt. Die Normale erwartet einen Vektor. Der *Vektor.YAxis* – Block beschreibt im Koordinatensystem einen Vektor vom Ursprung bis zum Punkt (0, 0, 1). Für die Variable *centerPoint* muss ein Punkt erzeugt werden. Punkte können entweder mit kartesischen Koordinaten oder mit Polarkoordinaten festgelegt werden. In diesem Fall handelt es sich um einen *Point.ByCoordinates* – Block, d.h. eine kartesische Beschreibung. Hier wird *x* und *y* über einen *Number* – Block definiert, während an y der Wert eines *Integer-Sliders* übergeben wird. Bei aktivierter Hintergrund-3D-Vorschau und einer automatischen Ausführung erscheint im Hintergrund der eben programmierte Kreis.

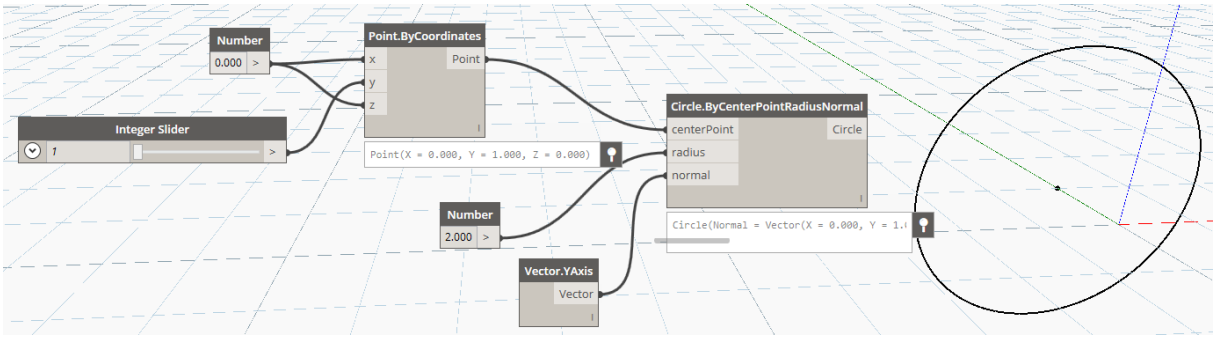

Abbildung 15: Anwendung von Blöcken und Verbinder an einem Beispiel

Oft werden zuerst bestimmte Geometrien erstellt, um mit ihnen andere Geometrien zu erzeugen. Durch die 3D-Darstellung aller Elemente wird das Modell irgendwann unübersichtlich. Beispielweise wird in Dynamo ein regelmäßiges Polygon mit Hilfe der Anzahl der Eckpunkte und einem Umfangskreis (Typ: *circle*) definiert. Für den Betrachter ist die zusätzliche Darstellung des Kreises nicht relevant. Da der Kreis anschließend nicht mehr gebraucht wird, kann in Dynamo mit einem Rechtsklick auf den *Circle* – Block die Hintergrund-3D-Vorschau deaktiviert werden.

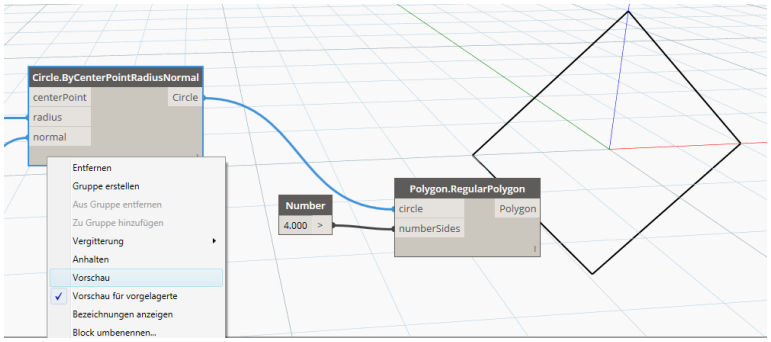

Abbildung 16: Dynamo-Funktion: Vorschau abwählen

4.2.3 Listen

Um ein Feld aus Zahlen wie in 3.1.1 zu erstellen, gibt es drei Wege. Der erste Weg ist mit dem Block *List.Create*. Für einfache Felder aus Zahlen ist dieser Block umständlich und nicht

automatisierbar. *List.Create* ist viel mehr beim Listen von Nicht-Zahlen interessant. Der Block besitzt eine Interaktionsmöglichkeit, mit der der Nutzer neue Indices für das Feld einfügen kann und somit gleichzeitig neue Eingabevariablen erzeugt. Der zweite und dritte Weg funktioniert über das Erstellen von Sequenzen. Einerseits ist die Erstellung von Sequenzen mit dem *Sequence* – Block und dem *Range* – Block möglich und andererseits über den einfachen aber textuellen Weg mit *Code Blocks*.

In Abbildung 18, Abbildung 19 und Abbildung 17 werden die drei Möglichkeiten zum Erstellen von Zahlenfeldern mit Hilfe des *Code Blocks* und dem *Range* – und *Sequence* – Block gegenübergestellt.

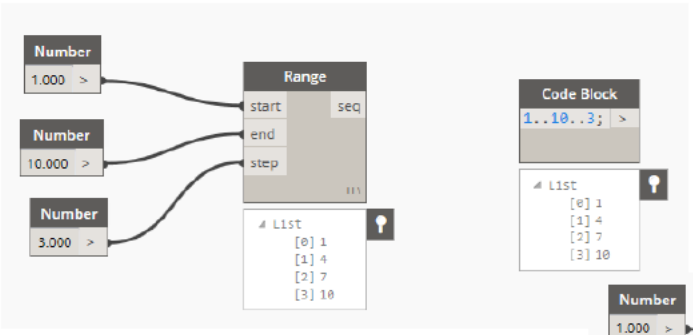

Abbildung 19: Vergleich des Range-Blocks mit einem Code-Block

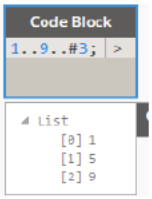

Abbildung 17: Weiter Möglichkeit der Sequenzdarstellung

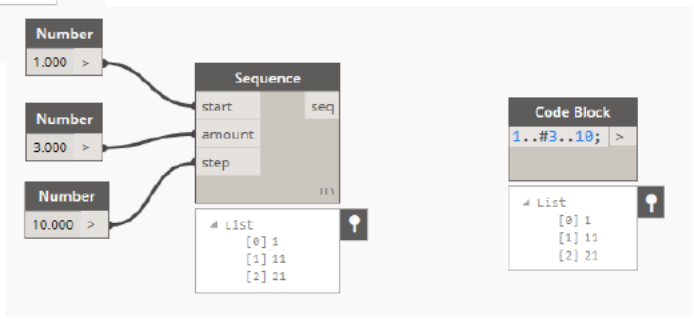

Abbildung 18: Vergleich des Sequence-Blocks mit einem Code Block

Wie bei vielen Programmiersprachen besitzt das erste Element einer Liste die Stelle (auch Index) 0. Durch die Raute wird die Anzahl der Elemente definiert. In *Code Blocks* können nicht nur Sequenzen erzeugt werden, sondern auch einzelne Elemente unterschiedlichen Typs. Zum Beispiel können Punkteblöcke mit den dazugehörigen Koordinaten wie folgt erstellt werden, ohne dass man weitere Blöcke und Verbinder braucht:

```
Point.ByCoordinates(0,0,1);
```

Da die Schreibweise im *Code Block* nicht schwer und übersichtlicher ist, wird diese textuelle Form trotzdem auch in einer visuellen Programmierumgebung bevorzugt.

Zahlenreihen sind beliebte und notwendige Mittel bei der Nutzung von Programmierungen. Werden Zahlenreihen beispielsweise mit der x- und y-Variable eines Point.ByCoordinates – Blocks verbunden und wird mit einem Rechtsklick die Vergitterungsart auf Kreuzprodukt geändert, können so Punktematrizen hergestellt werden. Diese Form der Vergitterung wird meistens für die Modellierung parametrischer Oberflächen hergenommen.

4.2.4 Benutzerdefinierte Blöcke

Beim Einsatz von wiederkehrenden Aufgaben können Blöcke in Dynamo zu sogenannten „Block-Gruppen“ oder „benutzerdefinierten Blöcken“ zusammengefasst werden. Dazu werden die benötigten Blöcke markiert und über ein Rechtsklick zu einer Block-Gruppe umgewandelt. Nachdem man einen Namen und die passende Kategorie für den neuen Block ausgewählt hat,

erscheint im Arbeitsbereich ein neuer Tab. Hier können die gruppierten Blöcke bearbeitet und Eingabe und Ausgabevariablen für den neuen Block geschrieben werden. In den *Input* – Blöcken stehen wahlweise die Namen der Variablen und die zu erwartenden Typen. Die Ausgabe wird über den *Output* – Block koordiniert, der ebenfalls mit einem Namen und einem Typ definiert werden kann. Beim Speichern eines Blocks kommt – im Gegensatz zu normalen Dynamo-Skripten – nicht das .dyn-Dateiformat zum Einsatz, sondern ein .dyf-Format. In Kapitel 4.3.2 wird der Einsatz von benutzerdefinierten Blöcken genauer veranschaulicht.

4.3 Auswertungsszenarien

In diesem Kapitel wird der Einsatz der visuellen Programmierung in einem BIM-basierten Modell analysiert. Dazu werden aus dem Bauingenieurwesen vier Fälle betrachtet, in denen Daten aus dem davor erstellten BIM-Modell herangezogen werden. Zum Erstellen des visuellen Programms wird Dynamo verwendet. Die einzelnen Beispiele umfassen unterschiedliche Bereiche im Bauingenieurwesen. Sie sollen in dieser Arbeit das Potential dieser Umgebung zeigen.

4.3.1 Quantitative Abfrage

Das Aufzählen eines bestimmten Bauteils in einem BIM-Modell ist mit Hilfe eines kurzen Skripts programmiert. Alle Bauteile sind vordefinierte bauspezifische Objekte. Diese werden von Revit als Familien abgespeichert und können an Dynamo weitergegeben werden.

allgemeine Mengenabfrage von Bauelementen

In diesem Skript findet eine Kommunikation mit dem Nutzer bei der Auswahl des Familienelements statt. Die Anforderung lautet:

Familientyp auswählen → Alle Elemente dieses Familientyps speichern → Feld zählen

Visuell programmiert sieht das Ganze wie folgt aus:

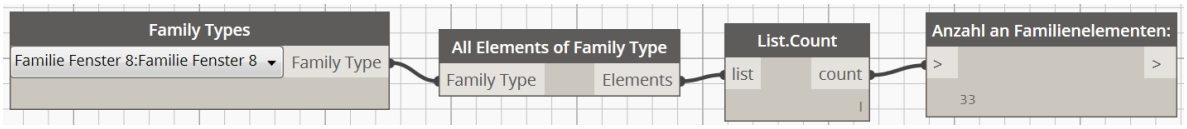

Abbildung 20: Allgemeine Mengenabfrage von Bauelementen

Der *Family Types* – Block speichert alle im Revit-Dokument enthaltenen Familien in einer Drop-Down-Liste ab. Aus dieser kann der Nutzer einen Familientyp wählen, woraus mit dem *All Elements of Family Type* – Block eine Liste aller im Projekt enthaltenen Familien erstellt wird. Die Liste enthält die Namen und die eindeutigen IDs der Familienelemente. Das Zählen der enthaltenen Elemente in diesem Array gibt die Anzahl der gewählten Familienelemente im Projekt zurück.

spezielle Mengenabfrage mit Hilfe von Kollisionserkennung

Interessanter ist die Mengenabfrage in einem bestimmten Bauteil. Beispielsweise: Wie viele Fenster enthält eine Wand. Obwohl die Beziehungen zwischen Bauteilen in einem BIM-Modell über inverse Attribute gespeichert werden, kann Dynamo diese Daten – zumindest ohne weitere Pakete – nicht herauslesen (Stand Version 1.0 Pre-Release). Deshalb wird hier, über die

geometrische Beziehung der Objekte im Koordinatensystem, die Abfrage ermittelt (vgl. Abbildung 21):

1 - Familienelemente zu einzelnen Festkörpern umwandeln

2 - Modelelement zu einem Festkörper umwandeln

3 - Schnittkörper erzeugen → Nicht erstellte Köper herausfiltern → Liste zählen

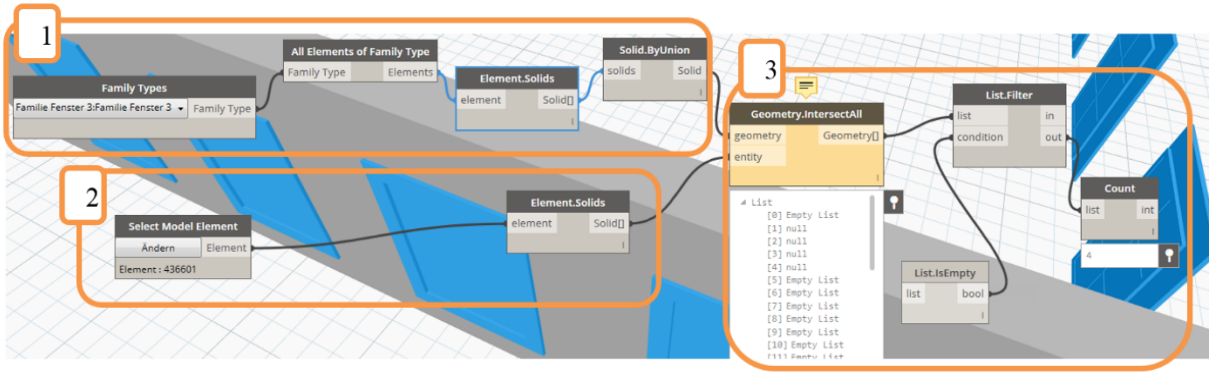

Abbildung 21: spezielle Mengenabfrage mit Hilfe von Kollisionserkennung

Ein Familienelement kann aus mehreren Extrusionen erstellt werden. Die Familie *Familie Fenster 3* besitzt beispielsweise 3 Extrusionen – Fensterrahmen, Glas, und Außenrahmen. Weil der *Element.Solids* – Block jede Extrusion einzeln betrachtet und so eine verschachtelte Liste erstellt, müssen die Unterlisten mit dem *Solid.ByUnion* – Block zu einem Solid zusammengefasst werden. Anschließend wird mit *Geometry.IntersectAll* jedes Fenster mit der Wand verschnitten und aus dem Schnittkörper eine Geometrie erstellt. Dieser Vorgang ist natürlich nur mit Fenstern möglich, die sich in der Wand befinden, weswegen Dynamo eine Warnung anzeigt. Für die Fenster, die keine gemeinsame Schnittmenge mit der Wand besitzen, legt *Geometry.IntersectAll* eine leere Liste an. Als dritte Möglichkeit ist gibt die Liste *null* wieder. Das ist in diesem Beispiel an den Stellen [1], [2], [3] und [4] der Fall. Das liegt daran, dass Fenster 3 aus drei Extrusionen besteht, die sich nicht alle schneiden. Der Außenrahmen ist beim Erstellen in Revit an der Wandaußenseite fixiert, während Glas und Fensterrahmen mit der Mittelachse der Wand verbunden sind. *List.Filter* erwartet eine Liste und filtert aufgrund einer booleschen Bedingung Elemente aus der Liste. Durch Zählen der Elemente in der gefilterten Schnittkörper-Liste wird anschließend die Anzahl des Familienelements in einem Modellelement bestimmt.

4.3.2 Aufbereiten von Parametern

Die Erstellung von Bauteillisten ist zur Dokumentation und Kalkulation eine wichtige Grundlage im Bauwesen. Es ist notwendig zu wissen, welche Arten eines Bauteiltyps im Projekt verwendet wurden und welche Parameter diese besitzen. Revit besitzt zwar ebenfalls eine Funktion für Bauteillisten, jedoch wird mit Hilfe von Dynamo einerseits der direkte Export in das Tabellenkalkulationsprogramm Excel und andererseits die individuelle Auswahl einzelner Elemente möglich. Außerdem kann mit Dynamo auch eine bereits erstellte Liste eingelesen und in einem BIM-Modell verwendet werden.

Ziel ist es, dass der Nutzer bestimmte Elemente aus dem Revit-Projekt in einer Excel-Liste schreibt. Man soll dabei die Möglichkeit haben, beliebige Parameter auszuwählen. Der Code wurde zum Nachverfolgen der Schritte in Abbildung 22 und Abbildung 23 dargestellt.

Nachdem eine bestimmte Kategorie ausgewählt wurde, werden mit *All Elements of Category* alle sich im Projekt befindlichen Kategorien eingeblendet. Nun soll der Nutzer die Elemente wählen können, die er in seiner Liste hinzufügen will. Hier ist anzumerken, dass Dynamo keine Aus- und Abwahlmöglichkeit anbietet. Stattdessen muss mit Hilfe eines Code-Blocks und einer *List.Create* eine Liste mit allen anzuzeigenden Elementen erzeugt werden. *List.GetItemAtIndex* gibt aus einer Liste eine definierte Stelle zurück. Der *List.Map* – Block liest jede Zeile einer an *list* übergebe Zeile und gibt sie jeweils an die erste unbelegte Eingabevariable des mit *f(x)* verbundenen Blocks weiter. Die *List.Map* enthält dann alle Elemente, die der Nutzer in seiner Liste einfügen will. Pseudocode 4 stellt die Funktion textuell dar.

```
function ListMap(elementnummer)
    for i=0:1:(length(elementnummer)-1)
        List.GetItemAtIndex(list,elementnummer(i))
    end
end
```

Pseudocode 4: Aufruf einzelner Elemente einer Liste durch List.Map

Mit *Element.Parameter* und *Parameter.Name* werden alle Parameternamen der gewählten Elemente gespeichert. Bei diesem Vorgang werden folglich Unterlisten erzeugt. *Flatten* addiert jede Unterliste auf und erzeugt eine 1-dimensionale Liste. Da jetzt in der Liste mehrere gleiche Parameternamen enthalten sind, müssen mehrfache mit *List.UniqueItem* herausgefiltert werden.

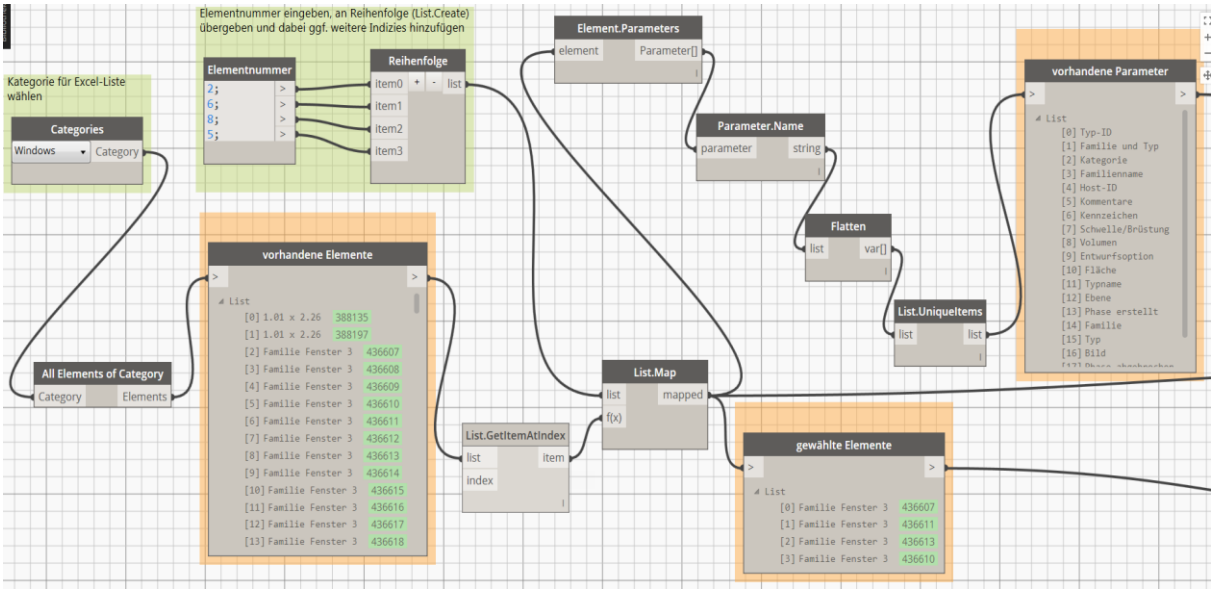

Abbildung 22: Skriptausschnitt zum Aufbereiten von Parametern – Auflisten vorhandener Parameter

Im zweiten Schritt soll der Nutzer aus den vorhandenen Parametern nach dem gleichen Prinzip gewünschte Parameter auswählen. Die Parameternamen der *List.Map* werden jeweils über eine weitere *List.Map* an *Element.GetParameterValueByName* übergeben. Dieser Block gibt bei Eingabe des Parameternamens den dazugehörigen Wert aus. Bei einer verschachtelten Liste ist es wichtig zu wissen, dass die Unterlisten zeilenweisen geschrieben werden. Die ersten Werte 3, 7, 9 und 6 stünden in einer Zeile. Sie müssen daher mit einem *List.Transpose* – Block transponiert werden. Die n*m-dimensionale Liste könnte jetzt direkt an *Excel.WriteToFile* übergeben werden. Jedoch soll schließlich auch erkennbar sein welches Element welchen Parameter besitzt.

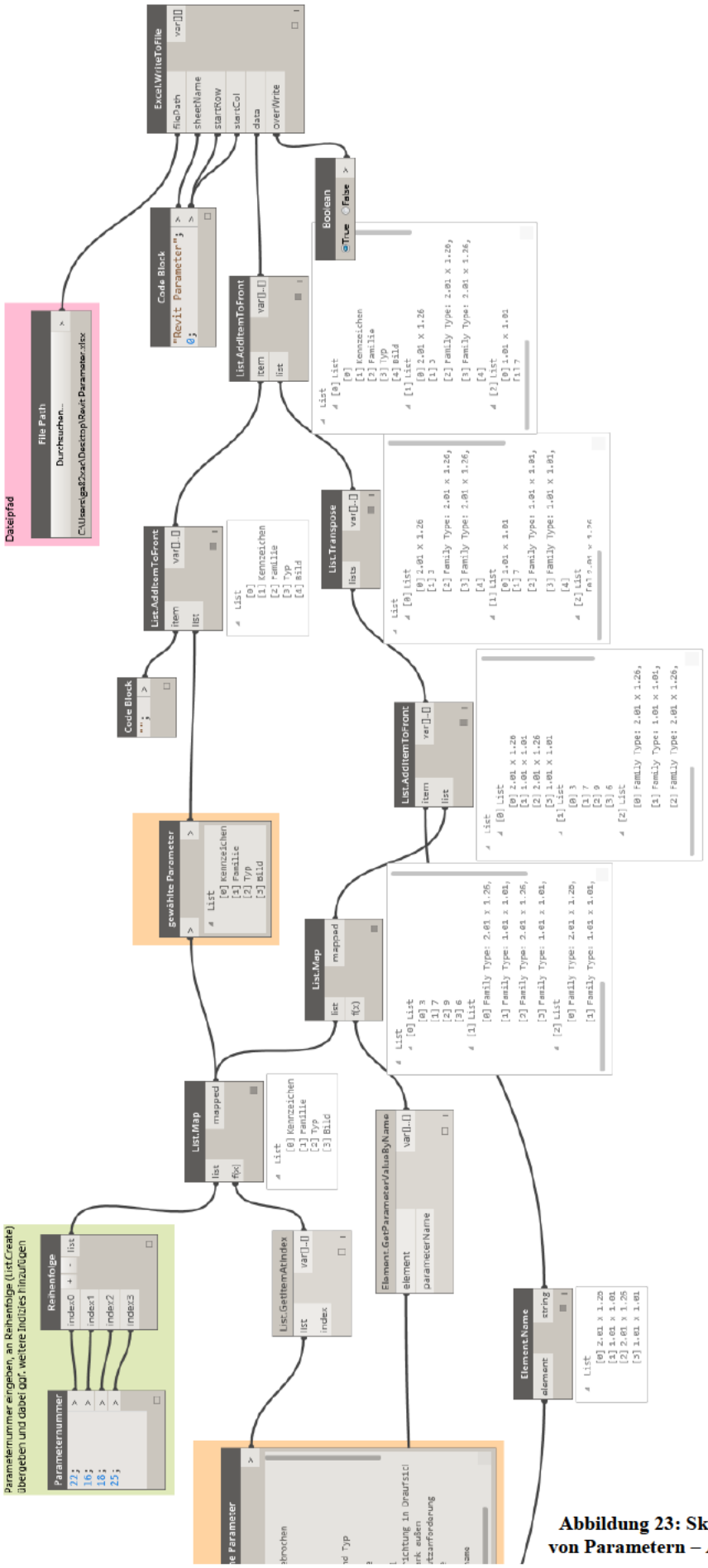

Abbildung 23: Skriptausschnitt zum Aufbereiten von Parametern – Aufbereiten der Daten für Excel

Daher wird vor der *List.Transpose* ein *List.AddItemToFront* – Block zwischengeschaltet. An die erste Stelle der Liste werden mit *Element.Name* die Namen der ausgewählten Elemente gebracht. Die Liste wird schließlich an oberster Stelle mit den Namen der ausgewählten Parameter ergänzt, wobei das erste Feld frei bleiben muss. Die Liste kann jetzt in einer Excel-Datei geschrieben werden. Der *Excel.WriteToFile* – Block erwartet einen Dateipfad, einen Tabellennamen, ein Startfeld in der Excel-Tabelle und natürlich die Liste. Die Eingabevariabel *overWrite* hat den booleschen Wert *false* als Vorgabe. Wird dieser so belassen, werden vorherige Einträge der Exceltabelle nicht gelöscht, bevor neue Werte hineingeschrieben werden.

	Kennzeichen	Familie	Typ	Bild
2.01 x 1.26	3	Family Type: 2.01 x 1.26, Family: Fenster 2-flg - Anschlag	Family Type: 2.01 x 1.26, Family: Fenster 2-flg - Anschlag	
1.01 x 1.01	7	Family Type: 1.01 x 1.01, Family: Fenster 1-flg - Anschlag	Family Type: 1.01 x 1.01, Family: Fenster 1-flg - Anschlag	
2.01 x 1.26	9	Family Type: 2.01 x 1.26, Family: Fenster 2-flg - Anschlag	Family Type: 2.01 x 1.26, Family: Fenster 2-flg - Anschlag	
1.01 x 1.01	6	Family Type: 1.01 x 1.01, Family: Fenster 1-flg - Anschlag	Family Type: 1.01 x 1.01, Family: Fenster 1-flg - Anschlag	

Tabelle 1: Ausgabe der Auswertung in Excel

4.3.3 Bauablaufsimulation

Ein BIM-Modell kann mit einem Soll-Terminplan abgeglichen werden. Einzelne Elemente des Modells werden dann mit einer Tabelle überprüft und die Abweichungen zwischen dem geplanten und dem tatsächlichen Bauablauf verglichen. So können für eine logistische Analyse Fortschrittskontrollen und Dokumentationen erstellt werden. Die Simulation der Solls erleichtert die Kommunikation mit Auftraggebern und Baubeteiligten. Sie kann den Bauablauf und die Zusammenhänge auf eine leichte, verständliche Weise sichtbar machen. Für Nicht-Bauingenieure und –Architekten ist es, im Vergleich zum Ablesen einer Ablauftabelle, einfacher und schneller geplante Vorgänge nachzuvollziehen.

In diesem Beispiel sollen beim Auswählen eines Bauteils in Revit alle zuvor zu erstellenden Bauteile chronologisch modelliert werden. Die Grundlage bilden die Parameter der einzelnen Elemente. Am Ende der Simulation wird in Dynamo ein 3D-Modell, mit den zuvor zu erstellenden Bauteilen erzeugt.

Weil das Beachten aller Fälle einen längeren Algorithmus erfordert, sind folgende Bedingungen zu beachten:

- Das Programm arbeitet unabhängig von eventuell definierten Phasen in Revit. Dennoch könnte auch für Phasen das gleiche Prinzip verwendet werden.
- Es wird davon ausgegangen, dass Wände, Tragwerksstützen und Decken aller unteren Ebenen vorhanden sein müssen, bevor bestimmte Bauteile erstellt werden
- Das Programm funktioniert bei der Auswahl von Wänden, Geschossdecken, Fenster, Türen oder Dächer. Im Einzelfall kann das Programm auch mit anderen Bauteilen problemlos laufen. Es kommt auf den jeweiligen Parameter an.
- Versatze werden nicht beachtet, sondern nur die Ebene oder die untere Abhängigkeit in der das Element erstellt wurde

Bedingungen

Mit einem *Select Model Element* – Block wird zunächst ein Bauteil aus Revit ausgewählt. Später sollen alle darunterliegenden Elemente modelliert werden. Daher ist die Lage des gewählten Elements zu ermitteln. Prinzipiell ist dies über die Überprüfung der geometrischen Lage möglich. Die Bauteile sollen jedoch stockwerksweise erzeugt werden, weswegen das Herauslesen der Ebene (Level) aus den Parametern für diese Simulation besser geeignet ist. Wie in 4.2 wird auch hier wieder ein *Element.GetParameterValueByName* – Block verwendet. Das Problem hierbei ist aber, dass der Parameter mit dem Wert vom Typ Level je nach Element unterschiedliche Namen besitzt. Zur Ermittlung der Level reicht es also nicht den Parameternamen „Ebene“ zu suchen, da dieser Name nicht in allen Bauelementen verwendet wird. Deswegen werden Fallunterscheidungen verwendet. Als erstes muss herausgefunden werden, um was für ein Element es sich handelt. Das geht wieder über den *Element.GetParameterValueByName* – Block. Hier wird nach der Kategorie gesucht. Aus der Ausgabe können nun if-Bedingungen gebildet werden.

Die Parameternamen aus den fünf Auswahlelementen mit einem Level als Wert sind wie folgt:

Kategoriename	Wände	Türen	Fenster	Geschossdecken	Dächer
Parametername mit Wert: Level	Abhängigkeit unten	Ebene	Ebene	Ebene	Referenzebene

Tabelle 2: Kategorie und Parameternamen der betrachteten Fälle für eine Bauablaufsimulation

Jetzt kann man den Code in Pseudocode 5 umsetzen.

```
P_name = „Kein unterstütztes Element gewählt“;
if K_name == „Wände“
      P_name = „Abhängigkeit unten“;
else if K_name == „Türen“ || K_name == „Fenster“ || K_name == „Geschossdecken“
      P_name = „Ebene“;
else if K_name == „Dächer“
      P_name == „Referenzebene“
end
Element.GetParameterValueByName(element, P_name);
```

Pseudocode 5: Erstellung von Bedingungen

Ein Ausschnitt des visuellen Codes ist in Abbildung 24 zu sehen.

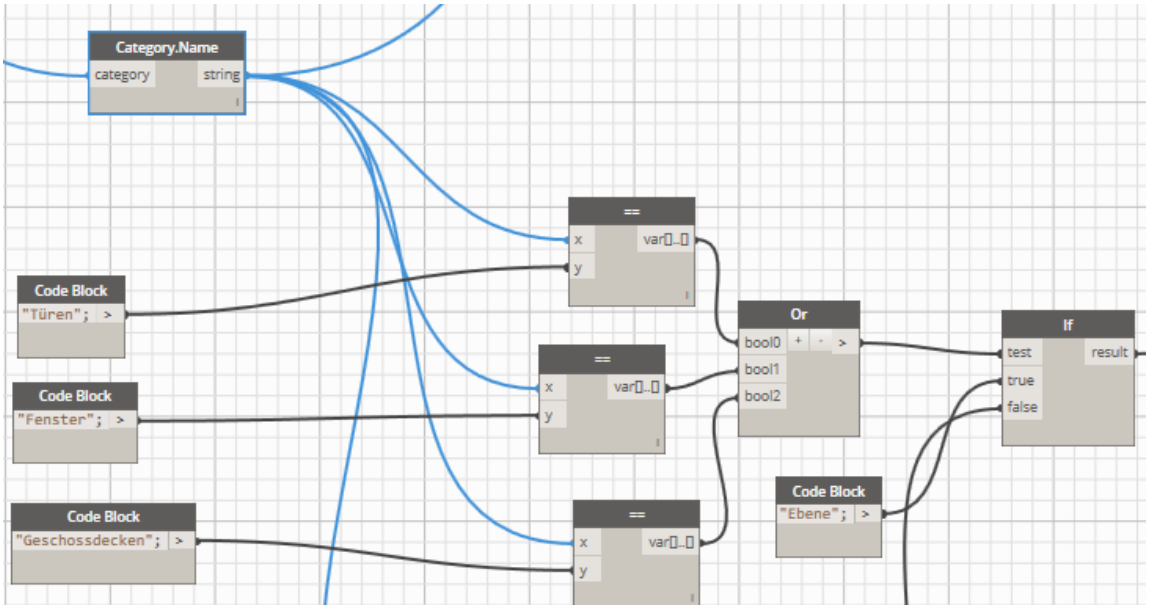

Abbildung 24: Ausschnitt von if-Bedingungen aus Dynamo

Anwendung von benutzerdefinierten Blöcken

Jetzt ist die Ebene des ausgewählten Elements ausgelesen. Als nächster Schritt sollen alle tiefer gelegenen Elemente bestimmt werden. In Dynamo gibt es die Funktion, um alle Elemente eines Levels aufzulisten. Man fängt daher am besten damit an, alle Levels herauszufiltern, die sich nicht unterhalb der gewählten befinden.

Levels sind in einem BIM-Modell ebenfalls als Kategorien gekennzeichnet. Mit dem *Category* – und dem *All Elements Of Category* – Block können alle im Modell vorhandenen Ebenen aufgelistet werden. Diese Liste ist aber zufällig angeordnet, und nicht nach der Höhe der Ebenen sortiert. In den Grundfunktionen von Dynamo gibt es einen *List.Sort* – Block. Dieser Block kann alphabetisch oder nach Größen sortieren. In diesem Fall würde die Sortierung von Levels nicht funktionieren, weil keine reinen Zahlengrößen in den Listen enthalten sind. Ein spezieller Code für die Sortierung von Levels ist erforderlich. Weil das Sortieren nach Level-Höhen später wieder gebraucht, wird zum Schluss der Sortier-Code zu einem benutzerdefinierten Block zusammengefasst. Die Idee beim Sortieren wird in Abbildung 25 gezeigt.

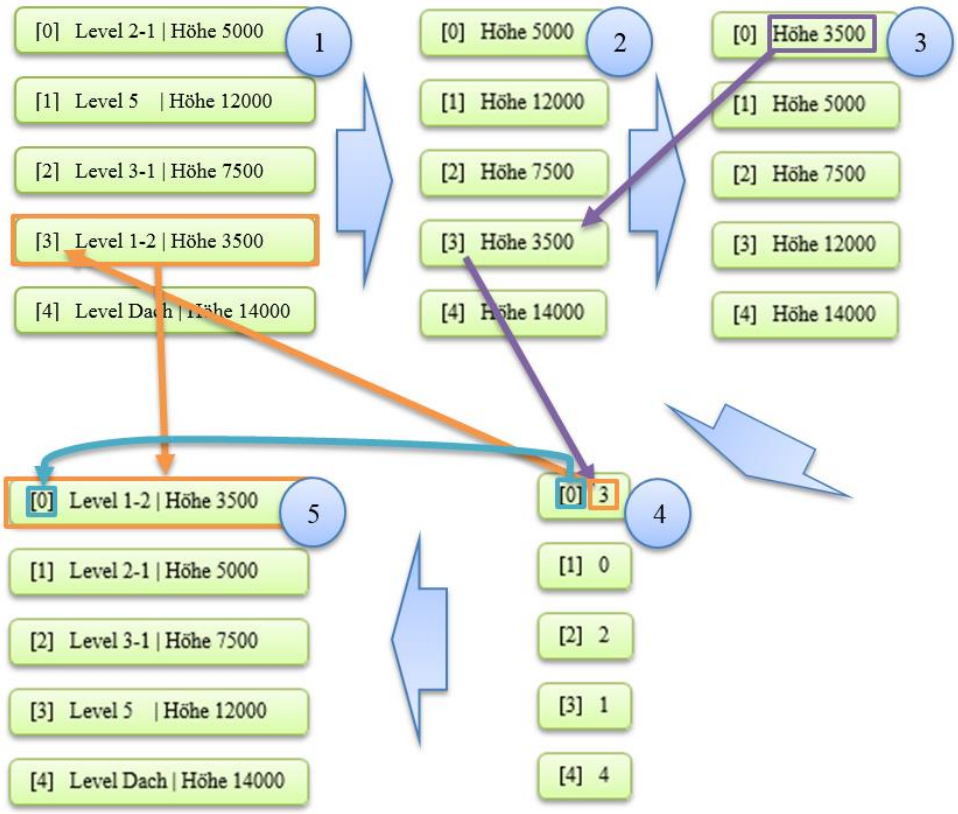

Abbildung 25: Gedankliche Schritte zum Erstellen eines Sortier-Codes

Die ungeordnete Liste bestehend aus Level-Name und Höhe ist die erste Liste. Mit dem *Level.Elevation* – Block wird dem Typ Level die Höhe aus 1 entnommen (Liste 2) und anschließend mit *List.Sort* der Größe der Höhen nach sortiert (Liste 3). Im nächsten Schritt wird mit dem *IndexOf* – Block aus der zweiten Liste nach den einzelnen Elementen der dritten Liste gesucht. Wenn das Element gefunden wurde, wird dessen Index in Liste 4 gespeichert. Auf diese Weise wird die Sortierfolge für Liste 1 erstellt. Dem *List.GetItemAtIndex* – Block werden als Index jeweils die Zahlen aus Liste 4 nacheinander übergeben. Anschließend wird der Inhalt eines Index aus der ersten Liste herausgeholt und somit eine nach Level-Höhen sortierte Liste 5 erstellt.

Die Schwierigkeit bei diesem Code liegt im letzten Schritt. Aus Liste 4 werden Position und Inhalt herausgelesen, um die sortierte Level-Liste zu erstellen. Der Schritt von Liste 4 auf Liste 5 wird hier genauer gezeigt. Aus Liste 4 braucht man die Inhalte und die Stellen aller Elemente. Zum besseren Verständnis wird zunächst nur mit dem ersten Element der Liste 4 gearbeitet. Aus Liste 4 wird also der Inhalt (*item* = 3) und mit *IndexOf* die Stelle (*index* = 0) herausgelesen. Im Code (Abbildung 26) werden dem *List.ReplaceItemAtIndex* – Block Liste 1, der Inhalt an der Stelle 3 der Liste 1 (mit *List.GetItemAtIndex*) und der Index 0 übergeben. Der Block gibt

dann eine Liste zurück, in der die Stelle 0 der Liste 1 mit dem Inhalt an der Stelle 3 der Liste 1 überschrieben wurde. Die so erzeugte Liste sieht wie in Abbildung 27 aus.

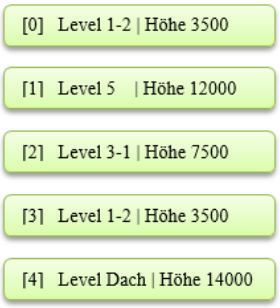

Abbildung 27: Erzeugte Liste beim ersten Element

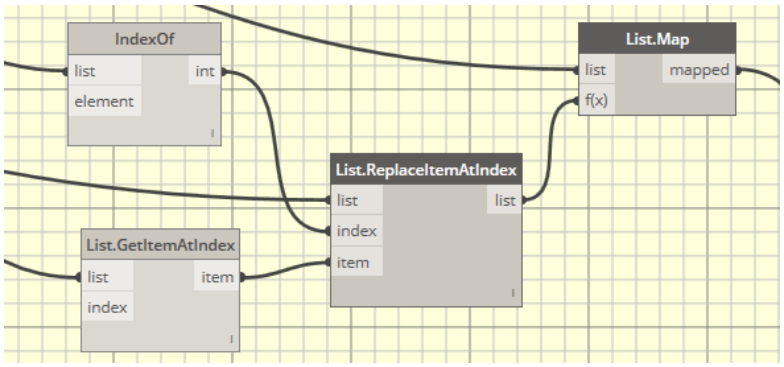

Abbildung 26: List.ReplaceItemAtIndex

Nun muss aber jedes Element der Liste 4 auf diese Weise ersetzt werde. Zum Durchlaufen einer Liste eignet sich wieder der *List.Map* – Block wie in Kapitel 4.2. Die List.Map gibt der ersten unbelegten Variablen nacheinander die Inhalte einer Liste und speichert die jeweiligen Ausgaben in einer neuen Liste. Das Problem in diesem Fall ist jedoch, dass der Inhalt der Liste 4 an zwei Eingabevariablen übergeben werden muss; einmal als gesuchtes Element an *IndexOf* und das zweite Mal als Index an *List.GetItemAtIndex*. Das Davorstellen und Verbinden eines Code-Blocks mit diesen Eingabevariablen führt zu einem Fehler in Dynamo. Das liegt vermutlich an der langen Verkettungsreihe. Aus diesem Grund müssen diese drei Blöcke zu einem definierten Block (namens *vertauschen*) gruppiert werden.

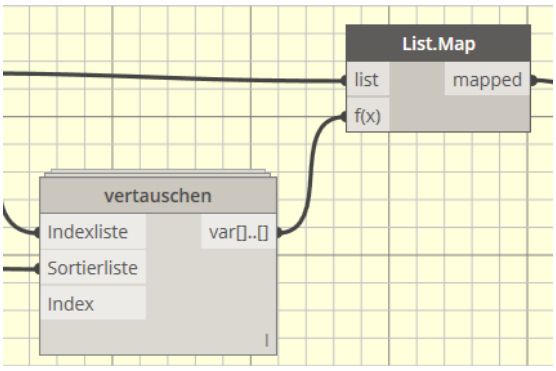

Abbildung 28: Benutzerdefinierter Block „vertauschen“

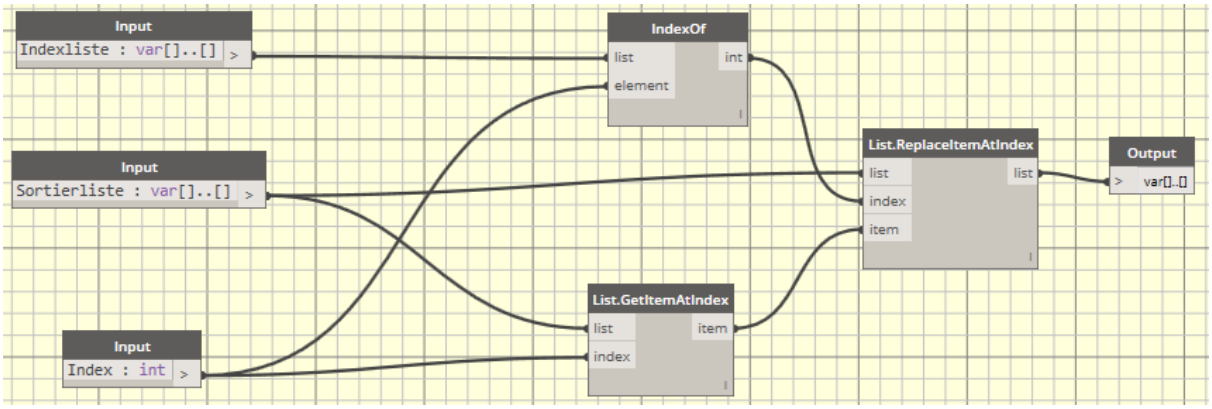

Abbildung 29: Inhalt des benutzerdefinierten Blocks „vertauschen“

Der *verstauchen* – Block enthält drei *Input* – Blöcke in der die Liste 4 als Indexliste, die Liste 1 als zu sortierende Liste und die Elemente aus der *List.Map* aufgenommen und verarbeitet werden.

Die List.Map hat bei jedem Element der Liste 4 eine Liste erzeugt, weswegen die *List.Map* aus mehrere Unterlisten in Form von Abbildung 27 besteht. Die sortierten Levels befinden sich an den Stellen $\sum_{k=0}^{n}[k,k]$, wobei n den letzten Index der Level-Liste darstellt. Um die Reihe in Liste 5 zu erzeugen, müssen diese Stellen herausgenommen und in eine Liste gepackt werden. Dafür erstellt man zwei *List.Map* – Blöcke in einem benutzerdefinierten Block namens *liste(n,n)*, die zusammen mit einem *Code Block* und einem *Count* – Block so ähnlich wie eine verschachtelte for-Schleife funktionieren. Der gesamte Sortier-Code wird anschließend als Block *SortLevelsByElevation* gruppiert.

Der nächste Schritt besteht darin, alle unter dem Level des ausgewählten Elements liegenden Level in einer neuen Liste einzufügen. Dazu wird ein *List.TakeItem* – Block verwendet. Dieser Block erwartet eine Liste und eine Anzahl. Aus der übergebenen Liste entnimmt er die ersten Elemente bis zur definierten Menge. Zusammen mit dem *IndexOf* – Block resultiert die Liste der Levels, die unterhalb des gewählten liegen. Alle Elemente in diesen Levels werden anschließend mit den bereits bekannten Blöcken aufgelistet. Alle Elemente eines Levels befinden sich dabei jeweils in einer Unterliste. Aus jeder Ebene werden mit dem benutzerdefinierten Block *FilterElementsByCategoryNames* Wände, Geschossdecken und Tragwerksstützen herausgefiltert. Das hat zur Folge, dass eine verschachtelte Liste (zu sehen in *List.Map*) entsteht, in der Unterlisten die Reihenfolge der Level-Höhen annehmen.

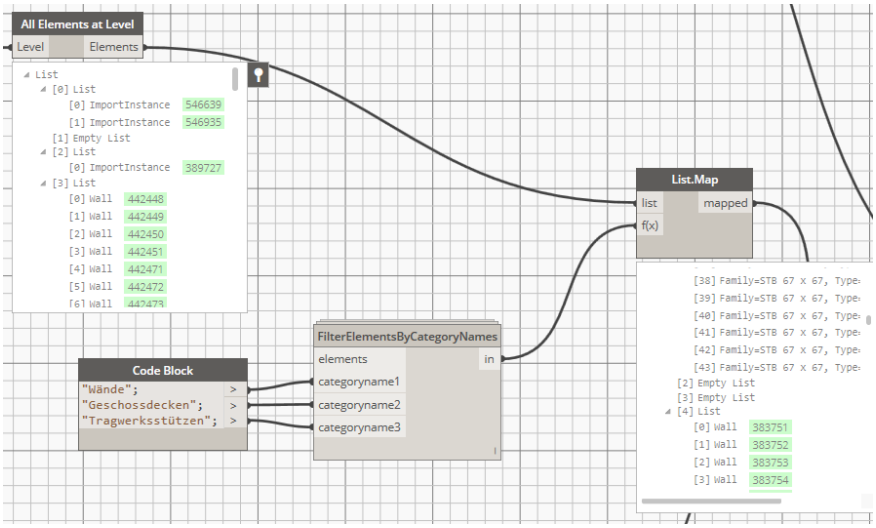

Abbildung 30: Benutzerdefinierter Block zum Filtern mehrerer Bauelemente

Bevor beispielsweise eine Wand modelliert wird, muss die darunterliegende Geschossdecke gebaut werden. Beim Erstellen von Geschossdecken in der Grundriss-Ansicht von Revit modelliert man die unter den Wänden befindliche Decke. Der Level der Geschossdecke ist dementsprechend die gleiche wie das ausgewählte Bauelement. Die Liste muss daher noch um die Geschossdecke erweitert werden. Diese Geschossdecke wird dann mit *List.AddItemToEnd* ans Ende der *List.Map* gebracht. Für den Fall dass ein Fenster oder eine Tür ausgewählt wird, muss logischerweise die dazugehörige Wand zuerst erstellt werden. Das heißt, dass diese Wand ebenfalls am Ende der Liste platziert werden muss. Zur Bestimmung der Wand werden alle Wände der gewählten Ebene auf die Möglichkeit eines Verschnitts mit dem gewählten Element überprüft. Die Wand, bei der die Überprüfung auf Verschnitt einen wahren Wert ergibt, wird an das Ende der Liste platziert. Beim Verwenden des *FilterElementsByCategoryNames* – Blocks ist es möglich, dass in manchen Ebenen im Revit-Projekt keine der drei Bauteile (Wände, Geschossdecken, Tragwerksstützen) vorhanden ist. Die dabei entstehenden leeren Listen (*Empty List*) müssen mit *List.Filter* aussortiert werden.

Somit ist die, vor dem gewählten Bauteil zu bauende Soll-Liste aller Elemente, inklusive ihrer Bau-Reihenfolge, erstellt. Diese werden anschließend im nächsten Schritt modelliert.

Simulation

Die hier verwendete Simulation ist keine Simulation im eigentlichen Sinne. Der Nutzer erhält die Möglichkeit die sortierten Listen mit Hilfe eines Schiebereglers stückweise zu modellieren. Bei jedem Bewegen werden mehr oder weniger Unterlisten der Soll-Liste entnommen. Als Schieberegler wird ein Number Slider verwendet. Es besitzt ein verstellbares Minimum und Maximum, zwischen denen man durch Verschieben double-Werte ausgibt. In diesem Fall die Werte zwischen 0 und 1 in Zehntel-Schritten. Mit Hilfe des Number Sliders und mathematischen Blöcken wird aus dem List.Filter - Block prozentuale Anteile vom Anfang der Liste entnommen und modelliert. Mit den Dynamo-Grundfunktionen ist ohne das Verwenden von Python-Skripten diese Methode die einzige und eine rechenintensive, um ein scheinbar bewegtes Bild zu erzeugen.

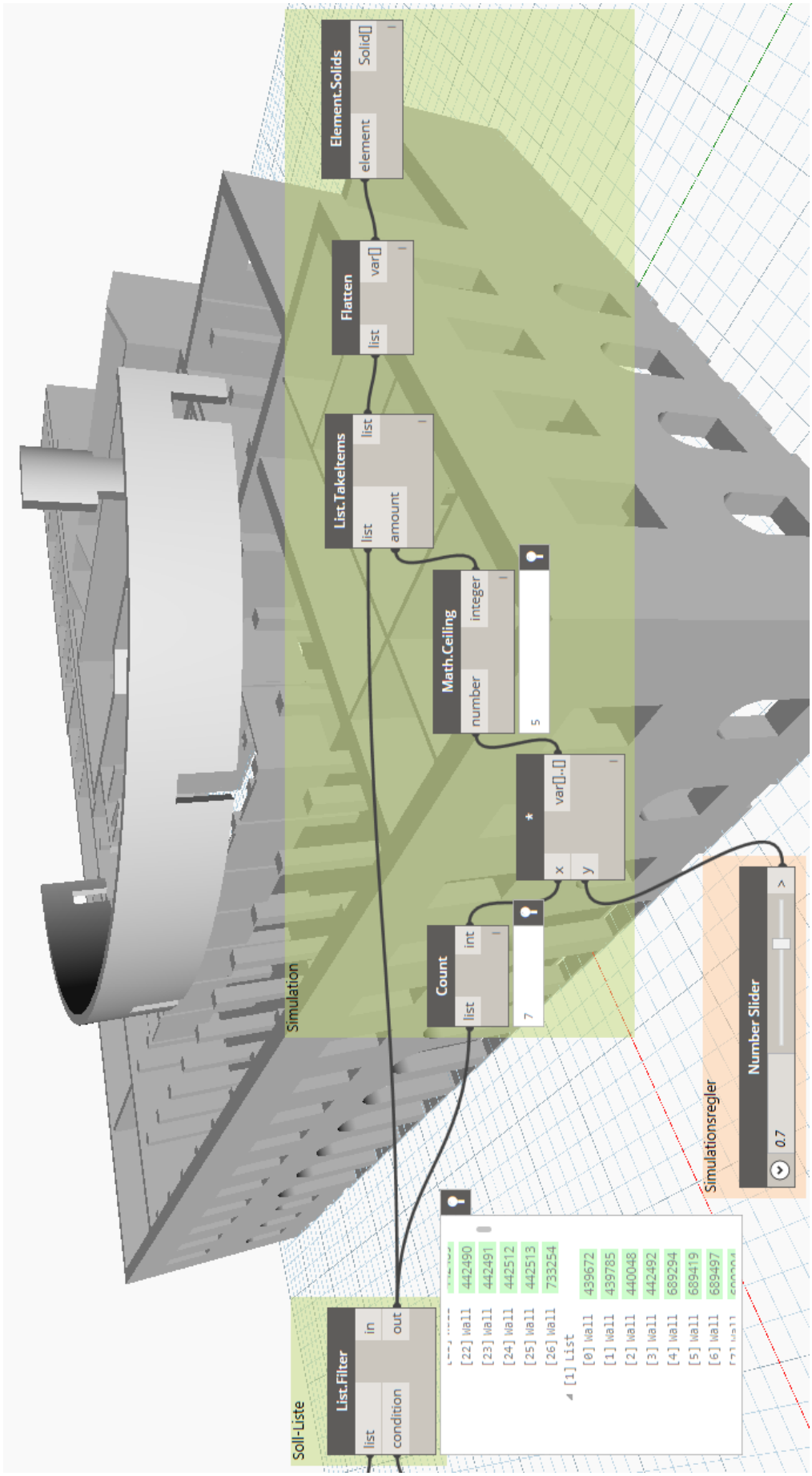

Abbildung 31: Ausschnitt der Simulation einer Bauablaufsimulation

4.3.4 Statisches System einer Decke

Mit Hilfe geometrischer Überschneidungen ist es in Dynamo möglich das statische System einer Decke zu modellieren. Hierfür muss in Revit vorher eine Modellinie generiert werden, aus der in Dynamo eine Schnittebene erstellt wird.

Die Vorgehensweise zum Erstellen von geometrischen Schnitten wurde bereits in vorangegangenen Beispielen vorgeführt. Deshalb werden nachfolgend nur die einzelnen Schritte nur kurz erklärt.

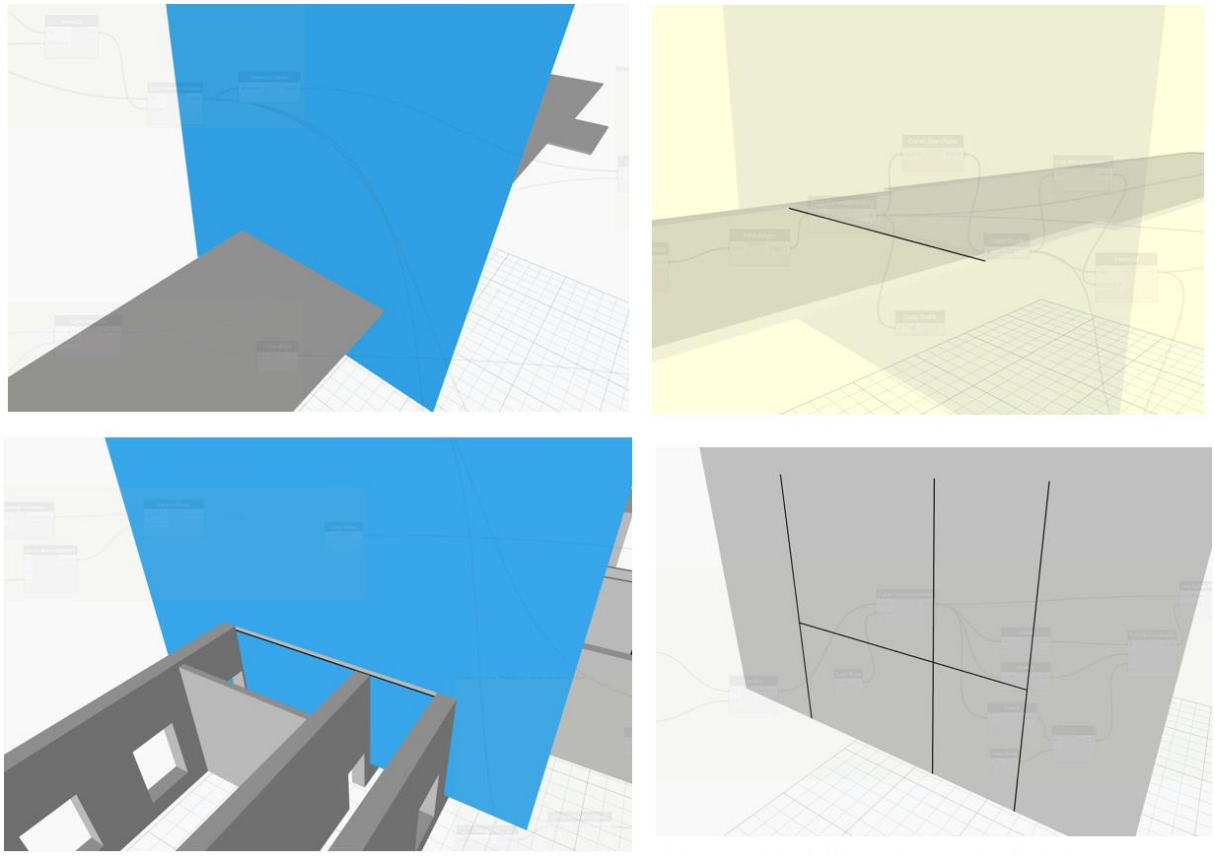

Abbildung 32: Vorgehensweise zur Erstellung statischer Systeme

Aus der Modellinie wird eine Kurve erzeugt und extrudiert. Daraus wird die Schnittfläche mit der gewählten Decke, gebildet. Aus dieser rechteckigen Schnittfläche wird dann mit Hilfe eines Codes (benutzerdefinierter Block: „*untere Kante von Rechtecksfläche*“) die untere Kante als Balken für das System ausgewählt. Ähnlich wie im Beispiel für die Bauablaufsimulation werden anschließend die Wände und Tragwerksstützen ausgewählt, deren obere Abhängigkeit die gleiche ist, wie die von der Geschossdecke. Das bedeutet, dass Wände mit einer manuellen oberen Abhängigkeit in diesem Beispiel nicht berücksichtigt werden. Die Decke und andere Elemente können für eine bessere Übersicht ausgeblendet werden.

Mit dem zuvor erstellten Block „*untere Kante von Rechtecksfläche*“ werden, aus der Schnittfläche zwischen der Wand und der Schnittebene, die unteren Kanten der Wände ausgewählt. Von den Mittelpunkten der Kanten werden dann senkrechte Linien nach oben gebildet. Die Schnittpunkte dieser Mittelachsen mit dem Balken ergeben die Stellen, wo sich Auflager befinden.

An diesen Punkten werden abschließend Auflager modelliert.

Abbildung 33: Ergebnis der Auswertung des statischen Systems

Man kann in den Parametern einer Wand deren Fähigkeit als tragendes Element abfragen und dementsprechend diese Wände für die Darstellung des statischen Systems beachten oder ignorieren. Für eine größere Entscheidungsfreiheit, wurde in diesem Beispiel eine Möglichkeit geschaffen, um einzelne Wände und Tragwerksstützen abzuwählen. Dafür wurden die einzelnen Auflager nummeriert (Rechtsklick auf den Block mit den Schnittpunkten → *Bezeichnungen anzeigen*).

Die Abstände einzelner Auflager und die Balkenlänge kann man herauslesen, um sie beispielsweise für Statik-Programme zu verwenden. Mit einem *DistanceTo* – und einem *List.Map* – Block wird ein benutzerdefinierter *DistanceBetweenPoints* – Block gebildet und so die Distanz zwischen benachbarten Schnittpunkte in der Liste errechnet. Die Entfernung des Balkens wird mit einem *Curve.StartPoint* –, einem *Curve.EndPoint* – und dem *DistanceTo* – Block ermittelt.

4.3.5 Weitere Auswertungsmöglichkeiten

Adjazenzmatrix aus Raumbeziehungen

Zur Bestimmung und Weitergabe der Zugänglichkeiten von Räumen eignet sich meistens eine quadratische n*n – Matrix, wobei n die Anzahl der Räume darstellt. Die Matrix besteht zunächst nur aus Nullen. Aus der Drop-Down-Liste des *Category* – Blocks werden Räume ausgewählt. Durch eine geometrische Überprüfung wird festgestellt, ob eine Tür sich zwischen zwei Räumen befindet. Die Räume, die eine Tür „umschließen“ haben die Möglichkeit der Begehbarkeit. Diese Stellen können in der Matrix mit 1 überschrieben werden. Die Matrix kann anschließend als .csv-Datei exportiert werden.

Exportieren

In Dynamo gibt es unterschiedliche Arten des Exports. Geometrische Modelle können als SAT-Datei exportiert werden und Ansichten als Bild oder als Rendering gespeichert werden.

Lichteinfallmenge in einem Raum

Mit dem *SunSettings.Current* – und dem *SunSettings.SunDirection* – Block können aus Revit der Sonnenstand und somit der Einfallsvektor der Sonnenstrahlen berechnet werden. Auf diese Weise lässt sich die Menge an Licht, das durch ein bestimmtes Bauteil (z.B. Fenster) hineinfällt berechnen. Bei ungenügendem Lichteinfall kann dementsprechend gehandelt werden.

5 Bewertung

5.1 Vor- und Nachteile der visuellen Programmierung

Die Stärken der grafischen Programmiersprache liegen in Anwendungsbereichen, in denen visuelle Metaphern umsetzbar sind (Schiffer, 2001). Daher liegt der größte Einsatzbereich der

VPLs im Bereich der Anfragesprachen (Ritter, et al., 2015). Ein allgemeiner Vergleich mit textbasiertem Code, lässt sich nicht treffen. In speziellen Gebieten, wie beispielsweise der Modellierung, kann der Einsatz virtueller Instrumente die Arbeit erleichtern. Bei typischen Aufgabenstellungen aus der Informatik ist die Programmierung mit VPLs nur eingeschränkt ausführbar, da auf die exakte Beschreibung des Prozesses verzichtet werden muss. Beispielsweise können Schleifen und rekursive Funktionen visuell kaum beschrieben werden. Daher ist Visual Programming nicht als Ersatz für die konventionelle Programmierug zu sehen, sondern vielmehr als nutzerfreundliche Umgebung für den Einstieg in die Welt der Informatik.

5.2 Visual Programming mit Dynamo

Anhand der Erkenntnisse während der praktischen Arbeit, werden im Folgenden die Funktionen von Dynamo für Revit bei bauingenieurstechnischen Aufgabenstellungen kritisch betrachtet. Unter anderem wird kurz auf die parametrische Modellierungsmöglichkeit der Software eingegangen.

5.2.1 Umsetzung und Mängel

Autodesk bietet mit Dynamo ein Programm an, dass die Einbindung von BIM-Modellen erlaubt. Die Arbeit mit Dynamo wird aufgrund vieler Beschreibungen und der strukturierten Bibliothek erleichtert. Außerdem unterstützt das Ausgeben von aktuellen Werten unterhalb eines Blocks, das Verständnis für den Prozess einer Ausführung. Die automatische Hintergrunddarstellung zeigt, ohne dass das BIM-Modell verändert wird, die modellierten Schritte und wird erst bei Bedarf auf das Hauptprogramm übertragen. All diese Hilfestellungen sprechen für die nutzerfreundliche Umsetzung der VPL in Dynamo.

Es gibt aber auch fehlende Funktionen, die man bei der Arbeit mit Building Information Modelling erwarten würde. Bei den Auswertungen der Szenarien werden aus dem Modell die Parameterwerte und die geometrischen Eigenschaften (Lage und Größe) einzelner Bauelemente herangezogen. Beim konkreten Fall „spezielle Mengenabfrage mit Hilfe von Kollisionserkennung“ gäbe es für die Abfrage der Bauteilbeziehung ein einfachere und fallsichere BIM-basierte Methoden. Ebenso beim Szenario „statisches System einer Decke“. Dynamo unterstützt jedoch in seinen Grundfunktionen keine Semantik (hier: Bauteilbeziehungen). Somit kann die visuelle Programmierumgebung ein grundlegendes Konzept, der ein großer Vorteil in der BIM-Lösung darstellt, nicht anwenden.

Dynamo ist nicht für die Erstellung von Endnutzer-Programmen ausgelegt, weswegen in dieser in den manchen Skripten umständliche *Code Blocks* und Beschreibung eingesetzt wurden. Die Umgebung wurde für den individuellen Gebrauch konzipiert.

5.2.2 Dynamo für parametrisches Modellieren

Dieser Eindruck fällt während der gesamten Nutzung von Dynamo immer wieder auf. Funktionen zur Bestimmung von Bauteil-Beziehungen kommen nicht zum Einsatz. Zur Demonstration des Einsatzgebiets „geometrische Modellierung“ wurden die Tische eines Hörsaals des modellierten Gebäudes mit Hilfe von Dynamo erstellt. Aus dem Revit-Projekt wurden vorhandene Geometrien und deren Lagen hergenommen und an ihnen geometrische Constraints angewandt. Die Tische wurden anschließend mit der Sweep-Methode entlang von mehreren Kurven erstellt.

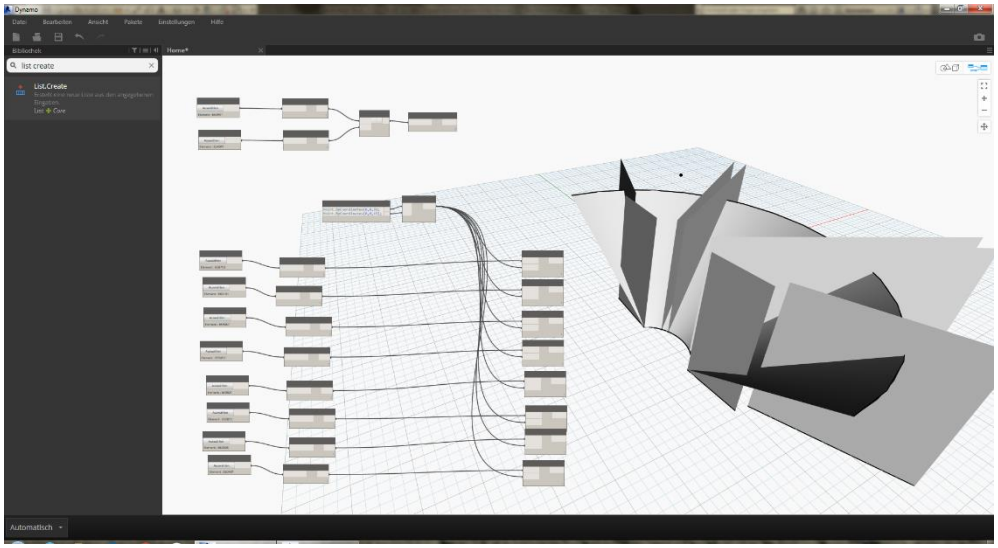

Abbildung 34: Modellierung von Tischen eines Hörsaals in Dynamo – Bildung parametrischer Abhängigkeiten

Abbildung 35: Modellierung von Tischen eines Hörsaals in Dynamo – Definition von Kurven

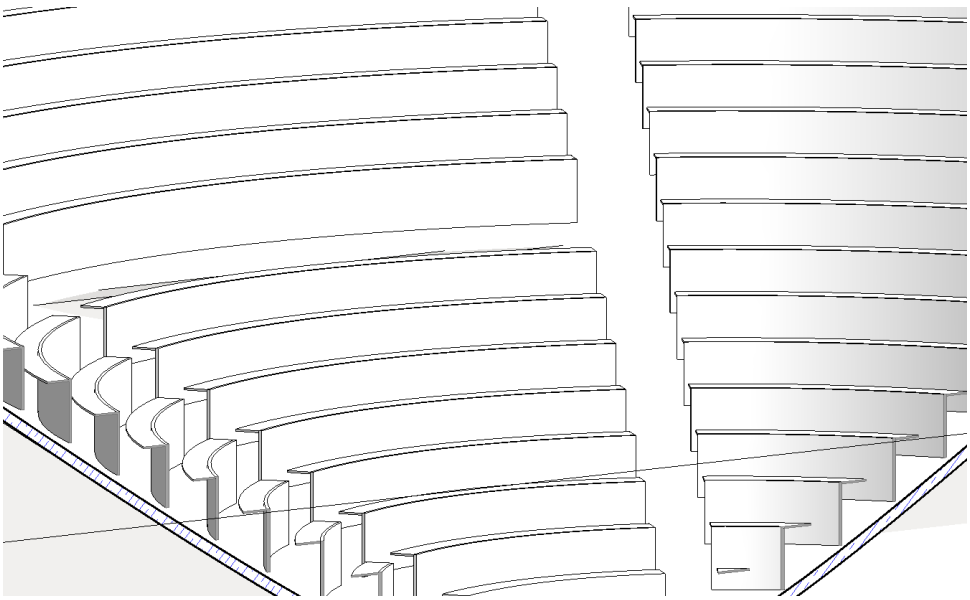

Abbildung 36: Aus Dynamo exportierte Tischmodelle in Revit

6 Fazit und Ausblick

Das Thema der visuellen Programmierung in computerbasierten Methoden ist ein vielversprechender Ansatz. Durch den Einsatz von grafischen Elementen wird ein Kommunikationsmittel geschaffen, das den Umgang mit Codes für Architekten und Bauingenieure erleichtert. Mit dem Erkennen des Potentials und der Anwendung der visuellen Programmierung in der Baubranche sind bereits die ersten Schritte getan. Jedoch kann man mit dem heutigen Stand der Technik, keine radikalen Veränderungen erwarten. Dynamo für Revit bietet die am weitesten entwickelte Software auf den Markt an. Aber die Software unterstützt eine wichtigste Eigenschaft vom BIM nicht – die Semantik. VPLs für BIM-Software existieren erst seit kurzem. Daher kann man davon ausgehen, dass sie visuelle Programmierung sich in der Zukunft durchsetzen wird.

Anhang A

Generierter Plan

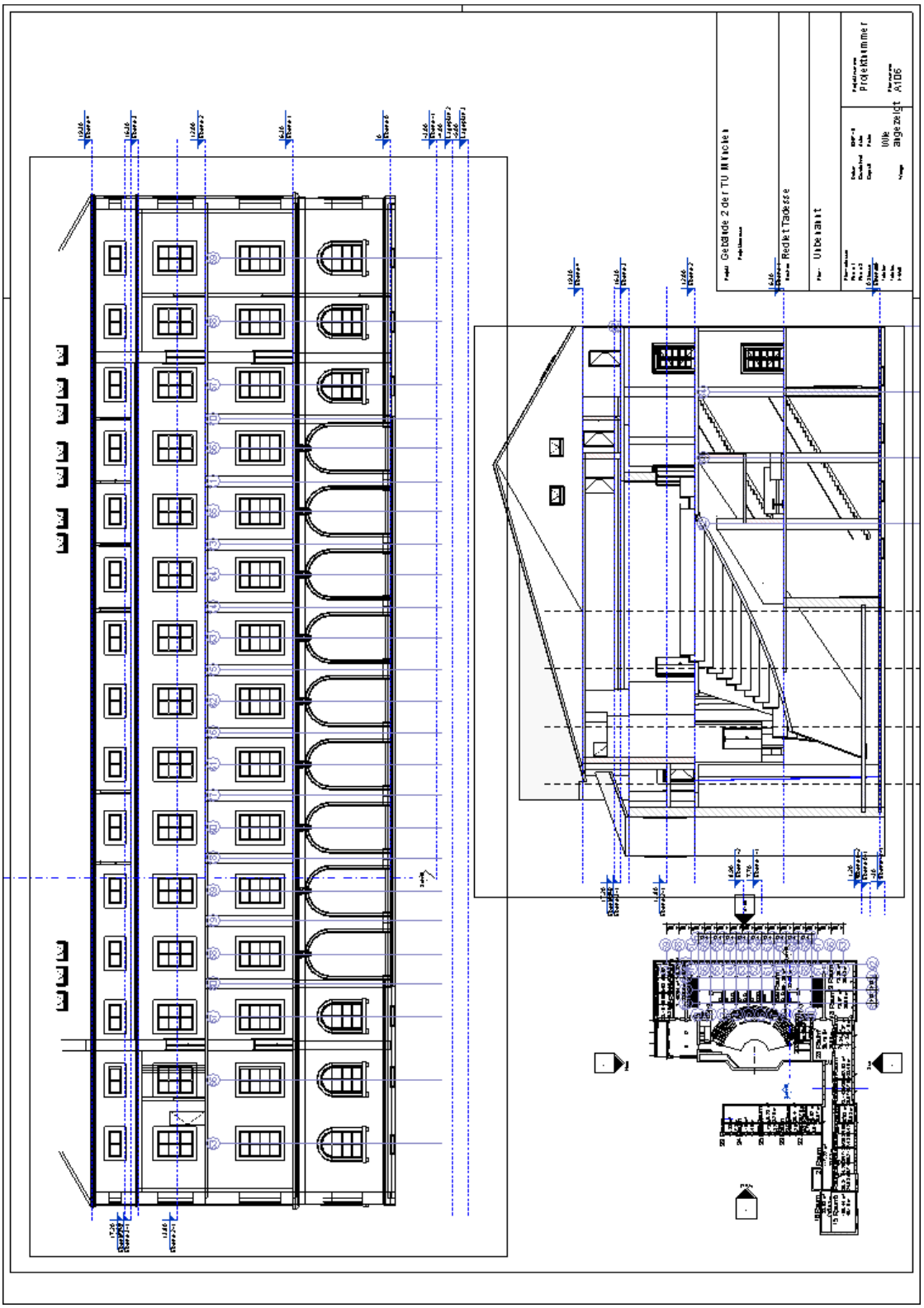

Abbildung A.0: Plan DIN A1 TU München Gebäude II

Anhang B

Digital Versatile Disc

Auf der beigefügten Digital Versatile Disc sind folgende Daten enthalten:

- Schriftlicher Teil der Arbeit als PDF-Dokument
- In dieser Arbeit beschriebene und erstellte Dynamo-Dateien
- In dieser Arbeit modelliertes Revit-Projekt

Literaturverzeichnis

Anon., 2016. *An example of a node-based algoritm in Grasshopper - Wikipedia.* [Online] Available at: https://upload.wikimedia.org/wikipedia/commons/f/f9/Grasshopper_NodeBasedSubtraction.png
[Accessed 20 Februar 2016].

Anon., 2016. *Lexikon online. duale Kodierung.* [Online] Available at: http://lexikon.stangl.eu/2505/duale-kodierung/ [Accessed 6 März 2016].

Berlin, H.-U., n.d. *Geometrische Modellierung von Objekten. CSG-Modell (Constructive Solid Geometry).* [Online] Available at: http://www2.informatik.hu-berlin.de/ki/lehre/ws0102/VLKogRob/Lexikon/Repraesentation/representation/csg.html [Accessed 6 März 2016].

Borrmann, A., 2015. *Objektorientierte Modellierung Datenaustausch mittels IFC.* [Online] [Accessed 27 Februar 2015].

Borrmann, A., König, M., Koch, C. & Beetz, Eds., J., 2015. *Building Information Modeling.* Wiesbaden: Springer.

Faas, T., Arzheimer, K. & Roßteutscher, S., 2010. *Information - Wahrnehmung - Emotion. Politische Psychologie in der Wahl- und Einstellungsforschung.* 1 ed. Wiesbaden: Springer.

Karen M. Kensek, L. B. A. A. & Douglas Noble, F., 2014. *Building Information Modeling.* New Jersey, USA: John Wiley & Sons, Inc.

Ritter, F., Preidel, C., Singer, D. & Kaufmann, S., 2015. *Visuelle Programmiersprachen im Bauwesen. Stand der Technik und aktuelle Entwicklungen.* [Online] [Accessed 27 Februar 2016].

Schiffer, S., 2001. *Visuelle Programmierung. Grundlagen und Einsatzmöglichkeiten.* Linz, Österreich: Addison Wesley Verlag.

Secerbegovic, L., 2015. *BIM me up!.* [Online] Available at: http://www.bim-me-up.com/die-zukunft-von-bim-ist-geskriptet/#more-277601 [Accessed 27 Februar 2016].

Abbildungsverzeichnis

Pseudocodeverzeichnis

Tabellenverzeichnis

Eidesstaatliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 7. März 2016

Rediet Tadesse

Rediet Tadesse
Dachauerstr. 95
D-80335 München
ga82xac@mytum.de