TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Erarbeitung eines Konzepts zur Definition von Gebäudemodellvariationen auf Basis von IFC-Dateien

An der Fakultät Bauingenieurwesen der Technischen Universität Dresden
zur Erlangung des akademischen Grades Diplom-Ingenieur für Bauingenieurwesen
genehmigte

Diplomarbeit

vorgelegt von

Ngoc Trung Luu
geboren am 7. Februar 1992 in Leipzig

ausgehändigt am 29. Juli 2015
eingereicht am 29. November 2015

verantwortlicher Hochschullehrer:
Prof. Dr.-Ing. Raimar J. Scherer

wissenschaftliche Betreuer:
Dipl. - Ing. Frank Opitz, Dipl. - Medieninf. Michael Polter

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Anlage 1

Aufgabenstellung für die Diplomarbeit Nr. 16

Name: cand. Ing. Ngoc Trung Luu

Vertiefung: Konstruktiver Ingenieurbau

Thema: **Erarbeitung eines Konzeptes zur Definition von Gebäudemodellvariationen auf Basis von IFC-Dateien**
(Elaboration of a concept for the definition of building model variations based on IFC files)

Zielsetzung:

Das Superpositionsprinzip verliert bei der nichtlinearen Tragwerksanalyse seine Gültigkeit, was dazu führt, dass für eine Vielzahl von Lastkombinationen eine separate nichtlineare Analyse durchgeführt werden muss. Die Definition der daraus resultierenden großen Anzahl verschiedener Parameterwerte erfolgt derzeit überwiegend in den softwarespezifischen Gebäudemodellen. Da während des Planungsprozesses i.d.R. eine Vielzahl verschiedener Programme zum Einsatz kommt, müssen diese Modellvarianten manuell an das jeweils softwarespezifische Format angepasst und in dieses übertragen werden.

Die Industry Foundation Classes (IFC), ein Standard zur digitalen Beschreibung von Gebäudemodellen (ISO 16739), dessen Nutzung in der Gebäude- und Projektplanung international zunehmend gesetzlich vorgeschrieben wird, stellen in ihrer jetzigen Fassung keine Konzepte zur Definition von Parametervariationen (z.B. Wertelisten, Wertebereiche, Formeln zur Berechnung von Werten) bereit. Da immer mehr Programme über eine Schnittstelle zum Import von IFC-Modellen verfügen, würde eine IFC-basierte Definition von Lastfall- und/oder Querschnittsvariationen die softwarespezifische Generierung diverser Planungs- bzw. Modellvarianten in der Tragwerksplanung enorm vereinfachen. Ziel der Diplomarbeit ist es, ein entsprechendes Konzept zu erarbeiten, zu testen und zu verifizieren.

Arbeitsumfang:

Während der Ausarbeitung sollen folgende Punkte bearbeitet werden.

1. Bewertung des Standes der Forschung und Technik im Bereich der variablen Belegung von Parametern in Gebäudemodellen
2. Untersuchung der Möglichkeiten zur Realisierung von Parametervariationen für IFC-Gebäudemodelle
    a. als Erweiterung von IFC (z.B. neue Klassen in EXPRESS oder neue Property Sets)
    b. in einem separaten Modell mit Verknüpfung zu den entsprechenden Elementen im IFC-Modell
3. Auswahl eines geeigneten Repräsentationskonzepts
4. Manuelle Verifikation des entwickelten Konzepts an einem Beispiel
5. Qualitative Bewertung von Grenzen, Einsatz- und Erweiterungsmöglichkeiten des entwickelten Konzepts

Wiss. Betreuer TU Dresden: Prof. Dr.-Ing. R. J. Scherer
Dipl.-Ing. Frank Opitz
Dipl.-Medieninf. Michael Polter

ausgehändigt am: 29.07.2015
einzureichen am: 29.11.2015

Prof. Dr.-Ing. R. J. Scherer
Verantwortlicher Hochschullehrer

Anlage 4

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsinstitution vorgelegt und ist auch noch nicht veröffentlicht worden.

Dresden, den

.....
(Unterschrift)

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Kurzfassung

Diese Arbeit befasst sich mit dem Austausch von Modellvarianten auf Basis von IFC-Daten. In der jetzigen Fassung stellt die Industry Foundation Classes (IFC), ein Standard zur digitalen Beschreibung von Gebäudemodellen (ISO 16739:2013), noch kein Konzept zur Definition von Gebäudemodellvarianten bereit, weswegen der Austausch von Modellvarianten mit großem Aufwand verbunden ist. Zur Lösung dieses Problems wird der Ansatz des Variationsmodells entwickelt.

Das Variationsmodell stellt ein Datenmodell dar, in dem sich Modellvarianten eines Gebäudemodells beschreiben lassen. Grundbausteine des Modells bilden dabei Parametervariationen, welche durch Variationswerte und Zeiger repräsentiert werden. Mithilfe der vom Modell bereitgestellten Verknüpfungsoperatoren lassen sich diese zu Modellvariationen zusammensetzen.

Die Generierung der im Variationsmodell beschriebenen Modellinstanzen erfolgt mit der Auswertung der Modellvariationen mithilfe einer entwickelten Software-Anwendung. Sowohl der Modellansatz als auch dessen Anwendung konnten an Beispielen erfolgreich verifiziert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

BIM Building Information Modeling

CAD computer-aided design

TGA Technische Gebäudeausrüstung

IFC Industry Foundation Classes

bSDD buildingSMART Data Dictionary

IDM Information Delivery Manual

XSD XML Schema Definition

XML Extensible Markup Language

ASN.1 Abstract Syntax Notation One

SGML Standard Generalized Markup Language

CSG Constructive Solid Geometry

GUI Graphical User Interface

UML Unified Modeling Language

1 Einleitung

Die modellbasierte Planung nimmt eine immer wichtigere Stellung im Bauingenieurwesen ein. Die dabei verwendeten Gebäudemodelle werden als Planungswerkzeug für die unterschiedlichen Gewerke genutzt. So finden diese unter anderem Verwendung in der Tragwerksplanung, Mengenermittlung oder in bauphysikalischen Simulationen. Während des Planungsprozesses entstehen i.d.R. eine Vielzahl von Modellvarianten in den softwarespezifischen Gebäudemodellen. Um diese auch in anderen Programmen nutzen zu können, müssen die Modellvarianten manuell an das jeweils softwarespezifische Format angepasst und in dieses übertragen werden. Der Vorgang ist daher mit einem großem Zeit- und Arbeitsaufwand verbunden. Mit den Industry Foundation Classes (IFC), ein Standard zur digitalen Beschreibung von Gebäudemodellen (vgl. [ISOa]), lassen sich Gebäudemodelle leicht zwischen den verschiedenen Programmen austauschen, insofern diese über eine entsprechende IFC-Schnittstelle zum Import von IFC-Modellen verfügen. In ihrer jetzigen Fassung bietet die IFC jedoch noch kein Konzept zur Definition von Parametervariation zum Austausch von Modellvarianten.

1.1 Motivation

Mit dem Konzept zur Definition von Modellvarianten soll eine effiziente Methode für die Erstellung und Speicherung von Modellvarianten geschaffen werden. Gegenüber der manuellen Variation des Modells mit dem Modellierungstool, soll der Ansatz der halbautomatisierten Modellgenerierung eine deutliche Verringerung des Arbeitsaufwands erzielen. Besonders wirksam kann dies bei Modellen sein, die für mehrere Anwendungen und Anwender genutzt werden sollen. Anstatt multiple ähnliche Modelle zwischen den Anwendern austauschen zu müssen, kann der Austausch allein mit dem Ausgangsmodell und dem Variationsmodell erfolgen. Beim Anwender werden aus diesen, mithilfe des entsprechenden Interpreters, die konkreten Modellinstanzen erzeugt. So können zusätzlich noch Speicherressourcen für den Datenaustausch geschont werden.

Besonders für Gebäudemodelle in IFC bietet sich diese Methode an. Als neutrales Austauschformat kann das Gebäudemodell und seine Varianten leicht zwischen den unterschiedlichen Software-Anwendungen ausgetauscht werden. Die verschiedenen Modellvarianten werden beim Anwender generiert und es können danach z. B. statischen Berechnungen, bauphysikalische Analysen, Terminplanung, Mengenermittlung etc. an den Modellen in der spezifischen Software vorgenommen werden, sofern eine IFC-Importschnittstelle oder ein Konverter für die Umwandlung vom IFC-Modell in das werkzeugspezifische Format existiert.

1.2 Ziele der Arbeit

Ziel der Arbeit ist es, ein geeignetes Konzept zur Definition von Gebäudemodellvarianten auf Basis von IFC-Dateien zu entwickeln und zu verifizieren, um den Austausch von Modellvarianten zwischen den verschiedenen Programmen, die im Planungsprozess zum Einsatz kommen, zu erleichtern.

Dazu werden in der Arbeit zunächst die Struktur und Elemente der Modellierungssprache EXPRESS sowie des Datenaustauschformats STEP vorgestellt, welche für das Verständnis der Arbeit notwendig sind. In einem zweiten Schritt werden bereits bestehende Ansätze der automatischen Modellgenerierung und Modellerweiterung in IFC untersucht. Darauf aufbauend wird im dritten Schritt das Konzept zur Definition von Gebäudemodellvariation auf Basis von IFC-Daten entwickelt, welches die Speicherung und Generierung von Modellinstanzen in einem Variationsmodell ermöglicht. Anschließend wird das Konzept an Beispielen verifiziert und die Software-Anwendung zur Generierung der Modellinstanzen aus dem Gebäudemodell und dem Variationsmodell erarbeitet. Zuletzt werden Grenzen und Erweiterungsmöglichkeiten des entwickelten Konzepts diskutiert und bewertet.

2 Grundlagen

In diesem Kapitel sollen die elementaren Grundlagen beschrieben werden, die zum Verständnis der nachfolgenden Kapitel notwendig sind. Es werden zunächst die Begriffe des BIM und der IFC kurz erläutert. Anschließend werden die Modellierungssprache EXPRESS und das Datenaustauschformat STEP vorgestellt, welche für das Konzept zur Definition von Gebäudemodellvariationen verwendet wurden.

2.1 BIM und IFC

Zu Beginn des Planungsprozesses eines Bauwerkes erstellt der Architekt traditionell einen 2D-Entwurfsplan mithilfe eines CAD-Programms. Dieser Entwurfsplan bildet die Grundlage für den weiteren Verlauf der Planung. Eine Kostenabschätzung zur Errichtung des Gebäudes wird durch eine Mengenermittlung auf Basis des Entwurfsplans erstellt. Im Verlauf der Planung kommen weitere Gewerke, wie z. B. Statiker, TGA Planer und Fachingenieure, hinzu. Sie alle nehmen mit ihren eigenen Fachplanungen Einfluss auf das gemeinsame Bauprojekt. Hierdurch entsteht ein Interessenkonflikt, der durch Kompromisse zwischen den Gewerken gelöst werden muss. Diese äußern sich in Änderungen der Pläne und sind im Planungsprozess unumgänglich. Da jedes Gewerk eigene Pläne für seine Arbeit nutzt, müssen bei einer Planänderung eines Gewerkes alle anderen Pläne der verbleibenden Gewerke abgeglichen und gegebenenfalls angepasst werden (s. Abb. 2.1). Es entsteht ein großer Koordinierungs-, Arbeits- und Kostenaufwand.

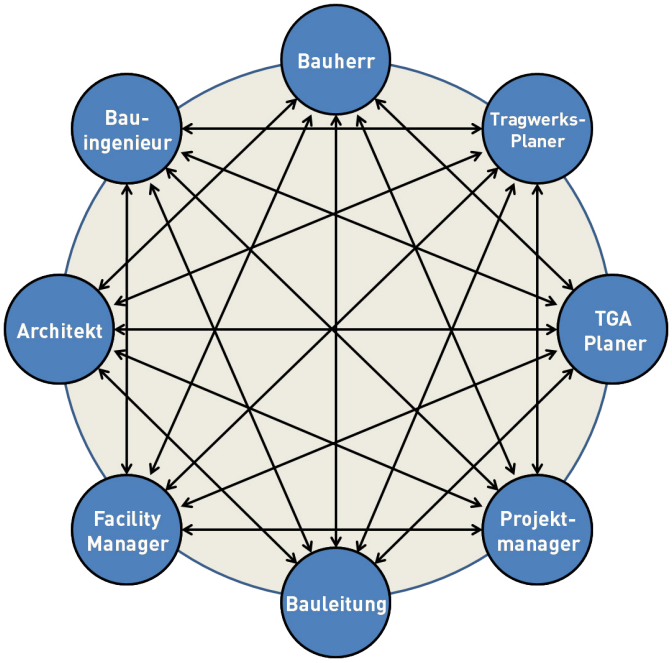


Abbildung 2.1: Klassischer Datenaustausch

Mit dem Building Information Modeling (BIM) sollen diese Probleme deutlich reduziert werden. Statt der traditionellen 2D-CAD oder der papierbasierten Praxis wird beim BIM ein 3D-Modell des geplanten Bauobjekts erstellt. Das Modell enthält neben den einfachen Geometriedaten des Bauwerks auch alle projektrelevanten Informationen (s. [EHLP13, S.18ff]). Als Beispiel soll ein Baustein dienen (vgl. Abb. 2.2). Unter Verwendung des BIM-Ansatzes werden sowohl die Bauteilgeometrie (3D-Modell) als auch die Informationen des Bausteins, wie z. B. Name, Farbe und Material, im Modell gespeichert. Bei der herkömmlichen Herangehensweise liegen nur 2D- bzw. 3D-Abbildungen des Bausteins vor.

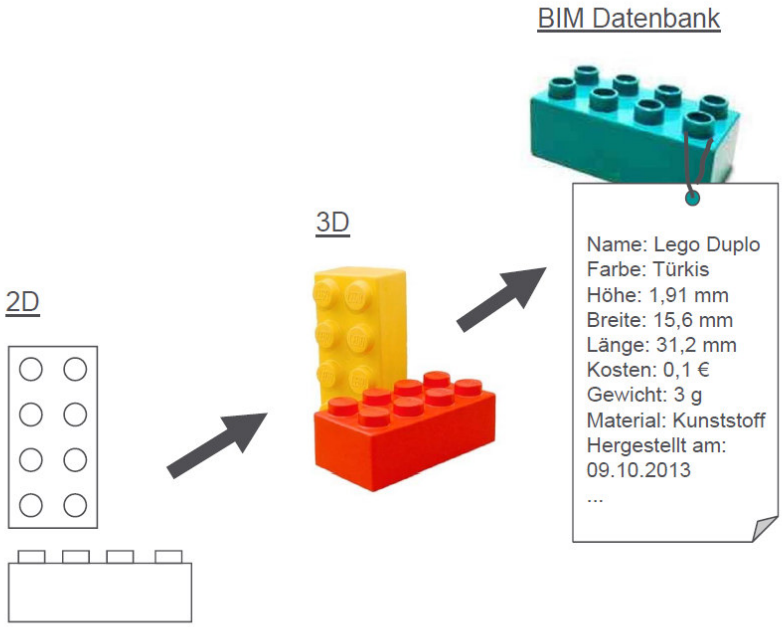


Abbildung 2.2: Daten eines Bausteins [Gre06]

Das Bauwerksmodell wird von den beteiligten Architekten und Fachplanern gemeinsam erarbeitet. Die Zusammenarbeit an einem einzigen Modell hat zur Folge, dass Daten immer aktuell und redundanzfrei vorliegen. Der Informationsaustausch zwischen den einzelnen Gewerken wird deutlich vereinfacht und potentielle Fehler, die beim Datenabgleich entstehen können, werden vermieden (s. Abb. 2.3).

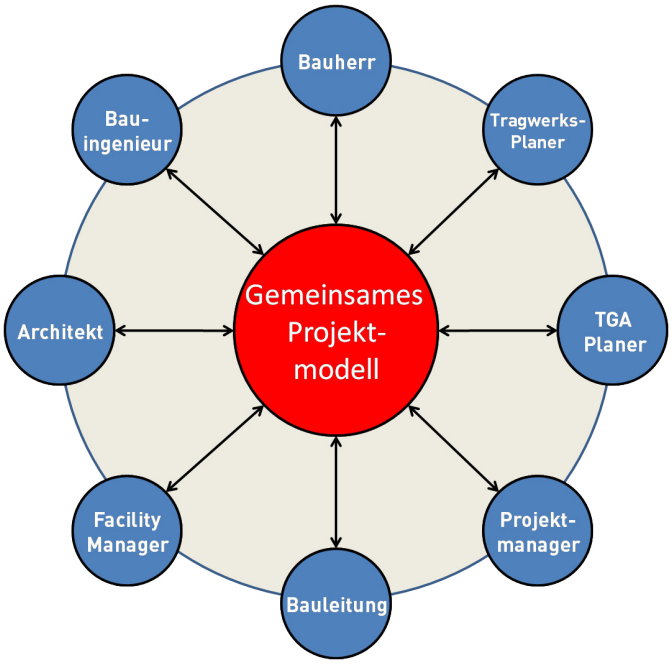


Abbildung 2.3: Datenaustausch unter Verwendung von BIM

2.1.1 buildingSMART

Die Organisation "buildingSMART international"(vgl. [Bui]) hat es sich zum Ziel gesetzt, die Planungsmethode BIM in der Baupraxis zu etablieren. Sie besteht aus einem Zusammenschluss von Hochschulen, Firmen und Privatpersonen aus dem Bauwesen und definiert Konventionen für den Datenaustausch zwischen den Projektbeteiligten, die für deren Vernetzung bei BIM notwendig sind. Dafür entwickelt buildingSMART offene Standards wie z. B.:

- **Industry Foundation Classes** (acsIFC): Dies ist ein Datenmodell für den Austausch von Bauwerksdaten zwischen proprietären Software-Anwendungen (s. Abschnitt 2.1.2).
- **buildingSMART Data Dictionary** (acsbSDD): Das Datenwörterbuch ist eine Referenzierungsdatenbank, die sprachenübergreifend Begriffe und Ausdrücke der Bauindustrie definiert und verknüpft.
- **Information Delivery Manual** (acsIDM): Die Regeln und Anforderungen für den Datenaustausch zwischen den beteiligten Akteuren werden im IDM festgehalten.

2.1.2 IFC

Den Grundstein für die Bestrebungen von buildingSMART bilden die IFC als digitales Austauschformat der Bauwerksdaten für die verschiedenen Software-Anwendungen der Projektbeteiligten. Als offener Standard[1] ist der Austausch möglich, sofern die entsprechenden Anwendungen eine Schnittstelle für IFC implementiert haben. Man unterscheidet bei IFC zwischen dem IFC-Datenschema, welches die Spezifikationen des Gebäudemodells definiert, und dem IFC-Modell,

[1]Die IFC sind seit dem Release IFC4 ein offizieller ISO-Standard - ISO 16739:2013

welches als Container für die Bauwerksdaten eines realen Gebäudes dient und zwischen den Anwendern ausgetauscht wird (vgl. [ISOa]). In den nachfolgenden Ausführungen wird sich auf die Version IFC4 (ehemals IFC2x4) vom März 2013 bezogen.

2.1.2.1 IFC-Schema

Die Spezifikationen für das IFC-Datenmodell werden im IFC-Schema definiert. Dieses beschreibt wie die Daten strukturiert werden und in welcher Relation sie zu anderen Daten stehen. Das Schema wird in der Datenmodellierungssprache EXPRESS formuliert. Daneben existiert seit der Veröffentlichung von IFC2x eine gleichwertige Spezifikation des IFC-Schemas in der XML Schema Definition (XSD) (s. [Lie09, S.18]). Die aktuelle Version des IFC-Schemas kann auf der offiziellen Webseite von buildingSMART gefunden werden. Genauere Informationen zum Aufbau des Schemas sind in [LT00] zu finden.

2.1.2.2 IFC-Modell

Im IFC-Modell werden die Daten eines spezifischen Bauwerks nach den Maßgaben des IFC-Schemas gespeichert. Es liegt für den Datenaustausch als Textdatei vor. Die Datei ist im STEP-Format geschrieben und besitzt die Dateiendung ".ifc". STEP steht für "**ST**andard for the **E**xchange of **P**roduct data" und ist ein genormtes Datenformat zum Austauch von Produktdaten. Zudem können die Bauwerksdaten auch im Extensible Markup Language (XML)-Format vorliegen.

Abbildung 2.4 zeigt die beiden Schemata und ihre zugehörigen Datenformate. Die Strukturierung des STEP-Formates erfolgt durch das EXPRESS- Schema. Entsprechend wird das XML-Format durch das XSD gesteuert. Im Folgenden wird die Variante des EXPRESS und STEP Physical File verwendet. Die entsprechenden Erläuterungen zu diesen sind in Abschnitt 2.2 kurz aufgeführt.

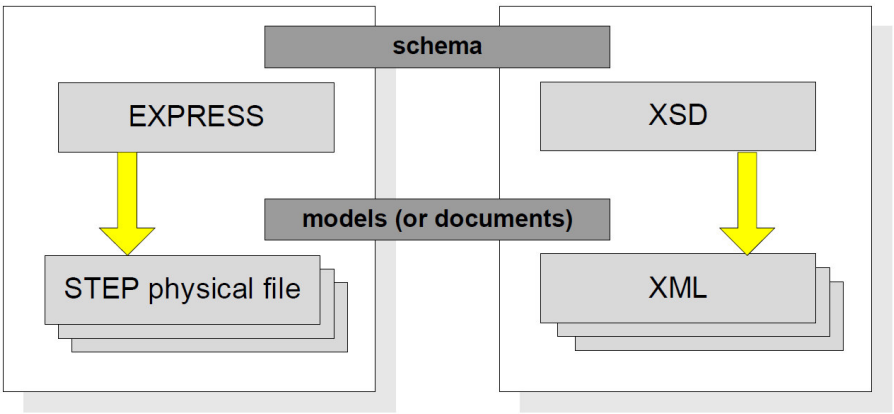


Abbildung 2.4: IFC mit EXPRESS und XSD [Lie09, Figure 1]

2.2 EXPRESS und STEP Physical File

Für ein besseres Verständnis für die folgenden Ausführungen wird in diesem Abschnitt eine kurze Erläuterung zu der Modellierungssprache EXPRESS und dem Datenaustauschformat STEP Physical File aufgeführt.

2.2.1 EXPRESS

Die EXPRESS-Sprache dient zur Datenmodellierung und ist unter der ISO 10303-11 [ISOb] definiert. Mit ihr lassen sich Spezifikationen von realen Objekten oder Prozessen in einer genormten Notation abbilden und werden für das IFC-Datenschema genutzt. Als Besonderheit gegenüber anderen Datenmodellierungssprachen besitzt EXPRESS sowohl eine graphische als auch eine textuelle Notation. Die graphische Notation wird als EXPRESS-G bezeichnet und bildet eine Untermenge zur textuellen Notation.

Die Strukturelemente der EXPRESS-Sprache sind Typendeklaration, Funktionen, Prozeduren und Regeln. Dabei bildet der Datentyp *Entity* das Kernelement der Modellierungssprache. Mit ihr können physikalische und konzeptionelle Objekte erstellt werden, die in der Regel in Relation zueinander stehen. Die Elemente der EXPRESS-Sprache werden im Folgenden kurz erläutert. Weitere Informationen zu Struktur und Aufbau von EXPRESS können aus [AT00], [Sch93] und der ISO-Norm [ISOb] entnommen werden.

2.2.1.1 Entitäten

Die *Entity* bzw. Klasse repräsentiert im Schema eine Vorlage für physikalische oder konzeptionelle Objekte. Sie entspricht der *Klasse* im objektorientierten Ansatz und wird durch Attribute und lokale Regeln (s. Abschnitt 2.2.1.4) definiert.

Attribute besitzen einen Attributnamen und beschreiben den Datentyp des Attributwertes. Sie liegen entweder explizit vor oder lassen sich aus den vorhandenen Attributen ableiten. Im zweiten Fall wird dieses Attribut unter dem Schlüsselwort *DERIVE* markiert und im Falle einer Instanziierung nicht als Parameter übergeben[2]. Weitere Modifikationen der Attribute können mit den Schlüsselwörtern *UNIQUE* und *OPTIONAL* vorgenommen werden. Durch *UNIQUE* muss die Wertebelegung des betreffenden Attributs für alle Instanzen des Entity-Typs verschieden sein. Instanzen, eines Entity-Typs mit einem *OPTIONAL* markiertem Attribut, müssen nicht zwingend mit einem Attributwert belegt werden.

Im Listing 2.1 wird die Entity *Baucontainer* deklariert. Sie besitzt die vier expliziten Attribute *Laenge, Breite, Hoehe* und *Inhalt*. Die ersten drei Attribute sind vom Datentyp *REAL*(reelle Zahl), während *Inhalt* ein Aggregationstyp von *Bauwerkzeug* ist. Zudem ist das Attribut optional, sodass eine Instanz der *Baucontainer* entweder keinen Inhalt oder eine oder mehrere

[2] Bei der Instanziierung werden die mit *DERIVE* gekennzeichneten Attribute aus den gegebenen Parameter berechnet und müssen daher nicht übergeben werden.

Instanzen von *Bauwerkzeug* enthält. Das Volumen, ebenfalls vom Typ *REAL*, ist ein abgeleitetes Attribut und wird aus *Laenge, Breite* und *Hoehe* berechnet. Die Entität *Bauwerkzeug* besitzt in diesem Beispiel keine Eigenschaften.

Listing 2.1: Beispiel Entity Baucontainer in EXPRESS

```
1  ENTITY Baucontainer;
2    Laenge: REAL;
3    Breite: REAL;
4    Hoehe: REAL;
5    Inhalt: OPTIONAL SET [1:?] OF Bauwerkzeug;
6  DERIVE
7          Volumen:REAL:= Laenge*Breite*Hoehe;
8  END_ENTITY;
9
10 ENTITY Bauwerkzeug;
11 END_ENTITY;
```

Entity-Typen können ebenfalls als Datentypen von Attributen verwendet werden, wodurch eine Beziehung zwischen den Entity-Typen bzw. deren Instanzen hergestellt wird. Im oberen Beispiel besitzt der *Baucontainer* eine 1:n-Relation zu *Bauwerkzeug*. Somit kann eine *Baucontainer*-Instanz eine oder mehrere *Bauwerkzeug*-Instanzen referenzieren. Durch ein inverses Attribut kann die Relation zwischen zwei Entities als bi-direktionale Beziehung gestaltet werden. Ein ”Bauwerkzeug” soll in diesem Beispiel entweder in genau einem ”Baucontainer” enthalten sein oder gar nicht. Es muss eine zweite Relation von *Bauwerkzeug* zu *Baucontainer* modelliert werden. Die Entity *Bauwerkzeug* wird dazu im Listing 2.2 mit dem inversen Attribut *Gehoert_zu*, modifiziert.

Listing 2.2: Beispiel Entity Bauwerkzeug in EXPRESS

```
1  ENTITY Bauwerkzeug;
2  INVERSE
3    Gehoert_zu: SET [0:1] OF Baucontainer FOR Inhalt;
4  END_ENTITY;
```

EXPRESS bietet für verschiedene Abstraktionsgrade das Konzept der Vererbung an. Damit lassen sich Entities generalisieren oder spezialisieren. Die vererbende Entity wird Supertyp genannt und vererbt all ihre Attribute und lokalen Regeln an die erbende Entity, dem Subtyp. Bei der Vererbung gelten folgende Regeln:

- ein Subtyp kann mehrere Supertypen besitzen (Mehrfachvererbung)
- ein Supertyp kann mehrere Subtypen besitzen
- ein Supertyp kann gleichzeitig auch ein Subtyp sein

- die Sub-/Supertypbeziehung ist transitiv[3]
- ein Supertyp kann nicht gleichzeitig ein Subtyp von sich selbst sein

Die vom Supertyp vererbten Attribute können im Subtyp modifiziert werden, sofern der zulässige Wertebereich des Attributs eingeschränkt wird. Der Subtyp ist damit immer spezialisierter als sein Supertyp. Neben der Modifikation der geerbten Attribute können auch neue Datenelemente hinzugefügt werden. Supertypen, die als reine Strukturelemente dienen, werden mit dem Schlüsselwort *ABSTRACT* als abstrakte Supertypen deklariert und können so nicht instanziiert werden.

Als Beispiel wird im Listing 2.6 die Entity *Produkt* als Supertyp der beiden Subtypen *Baucontainer* und *Bauwerkzeug* deklariert. *Produkt* ist hier die Generalisierung für ein kaufbares "Produkt". Sie wird als abstrakt deklariert, da es von ihr keine Instanzen geben kann und soll. Die Subtypen erben von dem gemeinsamen Supertyp die Attribute *Preis* und *Waehrung*.

Listing 2.3: Beispiel Entity Produkt in EXPRESS

```
1 ENTITY Produkt
2   ABSTRACT SUPERTYPE OF
3     (ONEOF(Baucontainer, Bauwerkzeug));
4   Preis: REAL;
5   Waehrung: Waehrung;
6 END_ENTITY;
7
8 % Ergaenzung bei den Subtypen
9 % "SUBTYPE OF (Produkt);"
```

2.2.1.2 Datentypen

Eigenschaften von Entitäten werden mithilfe von Datentypen beschrieben. Sie lassen sich in folgende Unterarten genauer klassifizieren:

- atomare Typen (*simple type*)
- Aggregationstypen (*aggregation type*)
- Auflistungstyp (*enumeration type*)
- Auswahltyp (*select type*)
- benutzerdefinierter Typ (*defined data type*)

Atomare Typen stellen die kleinste strukturierte Einheit dar. Sie können nur einen Wert des entsprechenden Wertebereichs annehmen. Die vorhandenen atomaren Datentypen sind in Tabelle 2.1 aufgelistet.

[3]Beispiel: Wenn A Supertyp von B ist, so ist B ein Subtyp von A

Tabelle 2.1: Simple Types in EXPRESS

Bezeichnung	Wertemenge
BINARY	0 oder 1
BOOLEAN	TRUE oder FALSE
LOGICAL	TRUE, UNKNOWN oder FALSE
INTEGER	ganze Zahlen
REAL	alle rationalen und irrationalen Zahlen
NUMBER	alle numerischen Werte
STRING	Zeichenkette beliebiger Länge

Aggregationstypen repräsentieren Gruppierungen von Elementen. Es existieren die Typen *ARRAY, LIST, SET und BAG*. Sie unterscheiden sich im Hinblick auf die Variabilität der enthaltenen Elementmenge, die Möglichkeit auf Dopplungen von Elementen und ob sie geordnet sind oder nicht. Die Eigenschaften der Aggregationstypen und die Syntax für deren Deklaration sind in Tabelle 2.2 aufgeführt.

Tabelle 2.2: Aggregation Types in EXPRESS nach [AT00, Tab. 4.3]

Typ	Syntax	Erläuterung
Feld	ARRAY [m:n] OF [Type]	Besteht aus genau n-m+1 Elementen, das erste Element wird mit m, das letzte mit n indiziert
geordnete, redundante Liste	LIST [m:n] OF [Type]	Enthält mindestens m und höchstens n Elemente. Die Elemente sind geordnet und dürfen mehrfach auftreten
geordnete, redundanzfreie Liste	SET [m:n] OF [Type]	Enthält mindestens m und höchstens n Elemente. Die Elemente sind geordnet und dürfen nur einmal auftreten
Menge	BAG [m:n] OF [Type]	Enthält mindestens m und höchstens n Elemente. Die Elemente sind nicht geordnet und dürfen mehrfach auftreten

Indem ein Aggregationstyp mit einem Entity-Typ als Attribut eines anderen Entity-Typs verwendet wird, lassen sich Relationen beliebiger Kardinalität, also m:n-Relationen, darstellen. Aggregationstypen besitzen nur eine Dimension, lassen sich jedoch untereinander verschachteln. So können beliebig viele Dimensionen mit den Aggregationstypen modelliert werden.

Der Auflistungstyp hat eine endliche Wertemenge aus verschiedenen Namen und muss innerhalb eines benutzerdefinierten Typs definiert werden. Ein Beispiel für den Auflistungstyp ist die Menge der Währungen (s. Listing 2.4). Die Wertemenge der Enumeration *Waehrung* kann z. B. den Euro, Dollar, Yen usw. enthalten.

Listing 2.4: Beispiel Auflistungstyp Waehrung in EXPRESS

```
1 TYPE Waehrung = ENUMERATION
2   OF(EUR, USD, JPY, CZK, GBP);
3 END_TYPE;
```

Der Auswahltyp ähnelt dem Auflistungstyp, bietet jedoch statt Namen eine Selektion von alternativen Datentypen (Entity-Typ oder benutzerdefinierter Typ) an. Auch hier muss die Definition innerhalb eines benutzerdefinierten Typs erfolgen. Listing 2.5 zeigt den Auswahltyp *Gegenstand*, welcher die Datentypen *Baucontainer* oder *Bauwerkzeug* anbietet.

Listing 2.5: Beispiel Selektionstyp Gegenstand in EXPRESS

```
1 TYPE Gegenstand = SELECT
2   (Baucontainer, Bauwerkzeug);
3 END_TYPE;
```

EXPRESS erlaubt über benutzerdefinierte Typen die Erweiterung oder Spezialisierung der Standardtypen. Dadurch ist es möglich anwendungsspezifische Datenstrukturen zu schaffen. So werden in IFC zum Beispiel die Datentypen *IfcAreaMeasure, IfcTimeMeasure, IfcMassMeasure* usw. definiert. Sie unterscheiden sich konzeptionell in ihrer Bedeutung (*IfcAreaMeasure* enthält Flächenwerte, während *IfcTimeMeasure* Zeitwerte repräsentiert etc.), jedoch ist der zugrunde liegende Datentyp, der durch den atomaren Typ REAL dargestellt wird, gleich.

2.2.1.3 Funktionen und Prozeduren

Mithilfe von Funktionen und Prozeduren können Algorithmen in ein EXPRESS-Schema eingebunden werden. Wie bei Programmiersprachen können die üblichen Kontrollstrukturen "Verzweigung" und "Schleife" sowie die "Zuweisungen" dafür verwendet werden. Es können zudem Parameter übergeben und lokale Variablen deklariert werden. Zur Generalisierung von Parametern können die Pseudo-Datentypen *AGGREGATE* und *GENERIC*, jedoch nur in diesem Kontext, verwendet werden. So ist *AGGREGATE* der Supertyp aller Aggregationstypen und *GENERIC* die Generalisierung aller verbleibenden Typen.

Eine Funktion besitzt einen konkreten Rückgabewert eines bestimmten Datentyps, den sie unter Verwendung der gegebenen Parameter ermittelt. Die Prozedur hingegen ist ein Algorithmus, der aus den übergebenen Parametern den gewünschten Endzustand bildet.

2.2.1.4 Regeln

Regeln sind ein weiteres Mittel um das Verhalten von Entitäten zu beeinflussen und schränken die Attributdeklaration zusätzlich ein. Man unterscheidet zwischen lokalen und globalen Regeln (s. [Sch93, S. 70]).

Lokale Regeln steuern das Verhalten der Attributwerte eines Entitäten-Typs. Sie können in zwei Gruppen unterteilt werden:

1. Die "*domain rule*" steuert das Verhalten von Attributwerten innerhalb einer Instanz eines Entity-Typs. Mit ihr lassen sich Rand- und Konsistenzbedingungen zwischen den Attributen einer Instanz abbilden. Als Beispiel wird im Listing die Konsistenzbedingung gesetzt, dass das *Preis*-Attribut der Entität *Produkt* nur positiv sein darf.

   Listing 2.6: Beispiel Entity Produkt mit Regel in EXPRESS

```
1 ENTITY Produkt
2 ABSTRACT SUPERTYPE OF
3 (ONEOF(Baucontainer, Bauwerkzeug));
4 Preis: REAL;
5 Waehrung: Waehrung;
6 WHERE
7  WR1: Preis > 0.;
8 END_ENTITY;
```

2. Die "*uniqueness constraint*" beschränkt die Belegung von Attributwerten innerhalb eines Entity-Typs. Sie kann in die "*simple uniqueness constraint*" und die "*joint uniqueness constraint*" unterteilt werden. Das "*simple uniqueness constraint*" verhindert das mehrmalige Auftreten von Werten des betreffenden Attributs und Entity-Typs; das "*joint uniqueness constraint*" verlangt, dass eine Wertekombination aus expliziten Attributen nur einer Instanz zugeordnet werden darf.

Für Spezifikationen zwischen verschiedenen Entity-Typen können globale Regeln formuliert werden. Ihre Deklaration ähnelt der von Funktionen und Prozeduren. Eine globale Regel bekommt die zu prüfenden Instanzen übergeben und prüft ob die in ihr spezifizierten Randbedingungen erfüllt sind.

In den vorangegangenen Beispielen sind Auszüge der Syntax und Semantik vom textuellen EXPRESS präsentiert worden. Für weiterführende Literatur zu EXPRESS können [ISOb], [And92], [AT00] hinzugezogen werden.

2.2.1.5 EXPRESS-G

Neben der zuvor beschriebenen textuellen Spezifikation bietet die Modellierungssprache EXPRESS zusätzlich eine graphische Notation zur Beschreibung eines Metamodells an. Abbildung 2.5 gibt einen Überblick über die grafischen Elemente von EXPRESS-G. Schemata können mit den gegebenen Symbolen visualisiert werden. Dabei ist anzumerken, dass die Visualisierung, im Gegensatz zur textuellen Notation, nicht das gesamte Schema aufzeigt. Regeln, Funktionen und Prozeduren werden in EXPRESS-G nicht abgebildet, jedoch kann mit der graphischen Notati-

on ein schnelleres Verständnis für Vererbungsbeziehungen und andere Relationen der Entities erhalten werden.

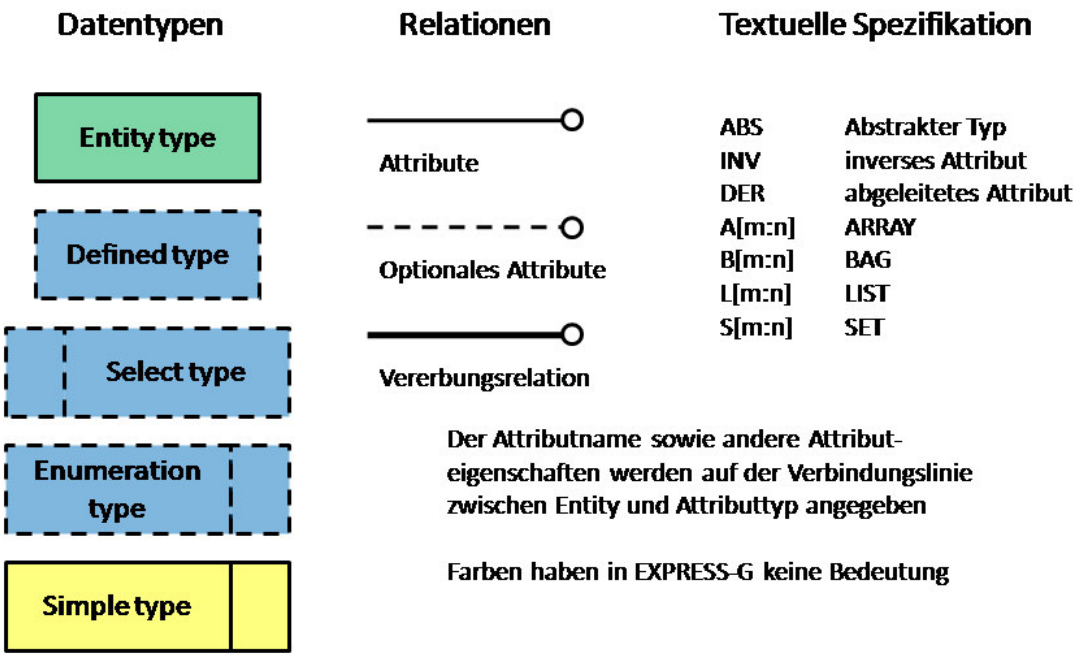


Abbildung 2.5: Überblick über die Hauptelemente in EXPRESS-G

Zur Veranschaulichung der Struktur eines Schemas in der graphischen Notation werden die in den vorangegangenen Beispielen deklarierten Entities und Typen (Listing 2.1, 2.2,2.6, 2.4 und 2.5) in Abbildung 2.6 mit der EXPRESS-G-Syntax dargestellt.

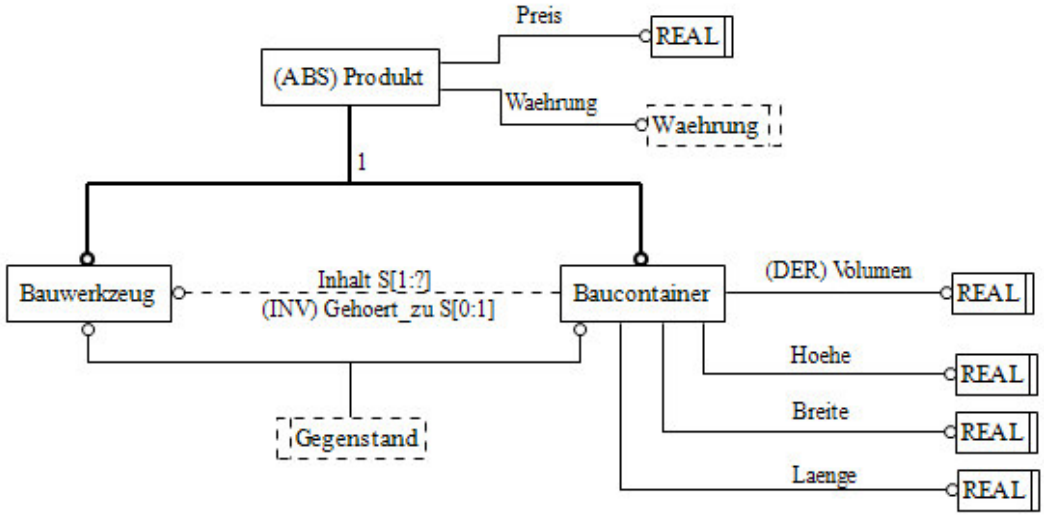


Abbildung 2.6: Beispiel Schema in EXPRESS-G

2.2.2 STEP physical file

Mithilfe von STEP lässt sich das IFC-Gebäudemodell in einer Textdatei abbilden. STEP ist ein internationaler Standard und ist unter ISO 10303-21 genormt [ISOd]. Durch die von STEP vorgegebene Struktur der Textdatei können die Bauwerksdaten plattformunabhängig eingelesen und geschrieben werden. Um den Daten einen Kontext zu geben, sodass diese von einer Software-Anwendung auch interpretiert werden können, muss eine übergeordnete Meta-Sprache verwendet werden. Ohne eine solche Meta-Sprache stellt die STEP-Datei nur eine lose Verknüpfung von Daten dar. Im Falle von IFC erhält die STEP-Datei über das IFC-Schema in EXPRESS ihren Informationsgehalt. Die STEP-Datei ist sequentiell gegliedert und setzt sich dabei aus zwei Segmenten zusammen:

1. **Header**: Im Header-Teil werden Metainformationen der Datei angegeben. Sie beinhalten beispielsweise den Dateinamen, den Namen des Erstellers und die genutzte Software, das genutzte Schema/Metasprache sowie das Erstellungsdatum.

2. **Data**: Der Data-Teil enthält die eigentlichen Gebäudedaten in der STEP-Datei. Dieser besteht aus den Instanzen der Schemaelemente.

Listing 2.7 zeigt das Grundgerüst einer STEP Physical File. Der Anfang und das Ende der Datei werden durch Schlüssel markiert. Die Segmente werden durch den Segmentnamen eingeleitet und enden mit einer ”ENDSEC;”-Zeile.

Listing 2.7: Aufbau einer STEP-Datei

```
1 ISO-10303-21;            /* Anfangsschluessel */
2 HEADER;
3 [ ... Header infomationen ... ]
4 ENDSEC;
5 DATA;
6 [ ... Entity Instanzen ... ]
7 ENDSEC;
8 END-ISO-10303-21;        /* Endschluessel */
```

Die Instanzen der Schemaelemente werden im Data-Segment hinterlegt. Sie werden in der STEP-Datei mit dem Namen ihres Schemaelements und den in Klammern gesetzten Attributen repräsentiert. Die Reihenfolge, in der die Attributwerte auftreten, ist im verwendeten Schema vorgegeben und gibt den losen Daten einen semantischen Gehalt. Jede Instanz erhält zudem einen Instanznamen in der Form ”#x”$\{x \in \mathbb{N}\}$. Mit diesen kann jede Instanz eindeutig identifiziert und referenziert werden, jedoch nur innerhalb der STEP-Datei.

Im Listing 2.8 ist ein Beispiel für die Abbildung eines Kreises zu sehen. Die Instanz von *IfcCircle* repräsentiert diesen Kreis und besitzt die zwei geordneten Attributwerte ”#9” und ”0.5”. Aus dem verwendeten IFC-Schema erschließt sich deren Funktion. ”#9” ist der Attributwert für die Lage des Kreises (*IfcAxis2Placement2D*) und ist eine Referenz auf das Objekt mit dem

selbigen Instanznamen. Über diese wird der Mittelpunkt des Kreises mit den Koordinaten [0.,0.] übergeben. Der Wert, ”0.5”, ist vom Typ *IfcPositiveLengthMeasure*, der für positive Maßangaben genutzt wird. Er gibt den Wert des Radius des Kreises an. Der abgebildete Kreis liegt demnach im Koordinatenursprung und hat einen Radius von 0.5 Längeneinheiten. Genauere Ausführungen zu STEP können aus [And92], [AT00] und der ISO-Norm [ISOd] selbst entnommen werden.

Listing 2.8: Instanzendeklaration in STEP

```
1 DATA;
2 #8=IFCCARTESIANPOINT((0.,0.));
3 #9=IFCAXIS2PLACEMENT2D(#8,$);
4 #10=IFCCIRCLE(#9,0.5);
5 ENDSEC;
```

3 Stand der Technik

Die Definition von Parametervariationen ist in der jetzigen Fassung der IFC nicht möglich. Ziel der Arbeit ist es daher, ein geeignetes Konzept für den Austausch von Gebäudemodellvarianten zu entwickeln. Dazu muss zum einen eine geeignete Datenstruktur für die Parametervariation in Form einer Schemaerweiterung gefunden werden. Zum anderen muss die Generierung der Modellvarianten aus den erstellten Parametervariationen und dem IFC-Gebäudemodell möglich sein. Im Folgenden werden verschiedene Ansätze der Schemaerweiterung in IFC an Beispielen untersucht. Zudem wird nachgewiesen, dass eine Modellgenerierung auf Basis von IFC-Daten möglich ist.

3.1 Schemaerweiterung mit IFC

Gebäude lassen sich unter Verwendung von IFC in virtuellen Gebäudemodellen beschreiben. Das IFC-Datenmodell kann jedoch nicht alle erdenklichen Gebäudedaten abdecken. Es existieren dafür zu viele Komponenten für die Konstruktion eines Gebäudes aus den verschiedenen Gewerken, welche berücksichtigt werden müssten. Grundsätzlich ist die Bildung von Klassen für alle Objekte möglich, jedoch würde die Implementierung des Modells immer aufwendiger und komplexer werden. Gleichzeitig kann damit eine "Verwässerung" des Schemas eintreten, d.h. dass definierte Klassen ihre repräsentative Eindeutigkeit verlieren. Deswegen werden neben der direkten Erweiterung des IFC-Schemas noch andere Möglichkeiten einer Schemaerweiterung mit IFC untersucht.

Die Form der Schemaerweiterung ist dabei stark von der Art der Beziehung zum ursprünglichen Schema abhängig. Die Verknüpfung der neuen Datenstrukturen an das IFC-Schema wird mithilfe von Links umgesetzt. *Link* ist dabei ein Überbegriff für einen Verweis auf Daten. Nach [Fuc15] lassen sich diese in Assoziationen, externen Referenzen, Relationsobjekte und Dokumentreferenzen unterscheiden. Im Folgenden werden die verschiedenen Linksysteme zur Erweiterung des IFC-Schemas betrachtet. Die Dokumentreferenzen entfallen hierbei aus der Betrachtung, da diese nur auf Dokumente verweisen. Dabei kann es sich um Texte, Bilder oder auch andere Fachmodelle handeln. Die Referenzierung ist dabei jedoch nur oberflächlich, d.h. dass keine Objekte innerhalb des Dokumentes referenziert werden, sondern nur das Dokument selbst. Die Verknüpfung der Parametervariation mit den entsprechenden Objekten des IFC-Gebäudemodells ist damit nur schwer umsetzbar.

3.1.1 Assoziation

Die Assoziation ist eine Referenz zwischen Dateneinheiten einer Modellinstanz, welche direkt vom Datenmodell vorgegeben ist. Die Umsetzung einer Schemaerweiterung über Assoziationen

entspricht daher einer direkten Erweiterung des IFC-Schemas. Ein Beispiel dafür ist der Vorschlag von [LK11] zur Aufnahme von Straßenbauelementen in die IFC. Es werden dabei neue Entities zur Beschreibung von Straßen-, Tunnel- und Brückenbauteilen erstellt, die in das IFC-Schema integriert werden sollen. Abbildung 3.1 zeigt die neu eingeführte Klasse *IfcRoad*. Sie repräsentiert das räumliche Element der Straße und kann sowohl in Längs- als auch in Querrichtung in untergeordnete Raumelemente unterteilt werden, in denen die physikalischen Elemente der Straße enthalten sind. Auf ähnliche Weise wird in [SA14] die Einführung von Straßenquerschnitten vorgeschlagen.

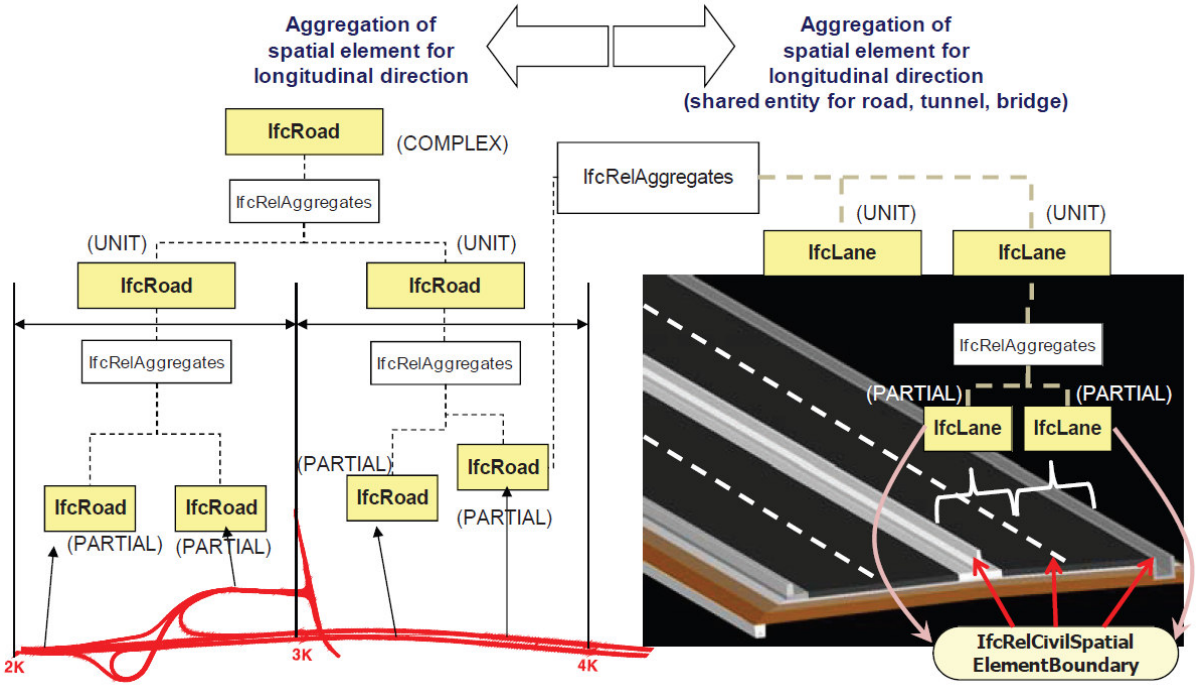


Abbildung 3.1: Beispiel für Aggregation räumlicher Elemente der Straße [LK11, Fig. 1]

Eine Umsetzung der Assoziation bedarf der Aufnahme des Variationsschemas in das IFC-Schema (vgl. Abb. 3.2), wodurch sowohl das Variationsmodell als auch das IFC-Gebäudemodell in einem gesamtheitlichen Modell vorliegen. Dabei bildet das Variationsmodell ein Teilmodell des Gebäudemodells. Das hat den Vorteil, dass bei Änderungen des Gebäudemodells die erstellten Parametervariationen nicht verfallen. Nachteilig bei dieser Lösung ist, dass für die Verwendung in anderen Software-Anwendungen eine Schnittstelle für das Variationsschema bereitgestellt werden muss. Zudem ist die Definition von Parametervariationen für die Beschreibung eines Gebäudemodells nicht essentiell, weswegen die Aufnahme des Variationsschemas in die IFC sehr unwahrscheinlich ist.

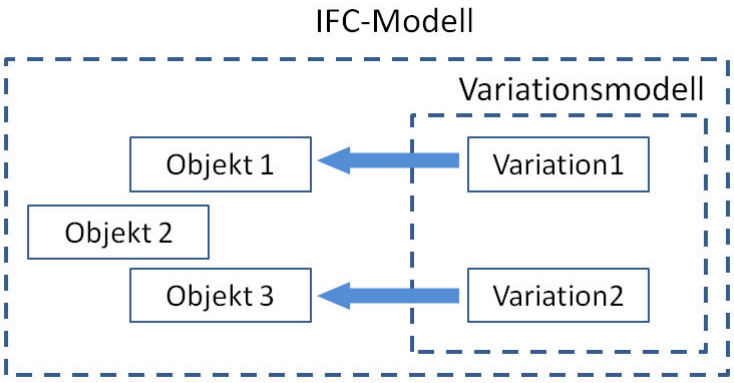


Abbildung 3.2: Variationsmodell mit Assoziation

3.1.2 Externe Referenzierung

Bei einer externen Referenzierung werden Dateneinheiten über die Modellgrenzen hinaus referenziert. Durch die externe Verknüpfung bilden die verbundenen Fachmodelle einen geschlossenen Informationsraum. Dabei existiert in der Regel ein führendes Fachmodell, welches Eigentümer der externen Links ist. Das führende Modell ist damit abhängig von den referenzierten Fachmodellen und somit an deren Lebenszyklus gebunden. In diesem Zusammenhang wird von [KFB05] eine IFC-Erweiterung vorgeschlagen, welche eine Referenzierung zwischen IFC-Modellen erlaubt.

Bei einer Einführung eines Variationsmodells mithilfe von externen Referenzierungen nimmt dieses die Rolle des führenden Modells ein. Die Variationen des Variationsmodells werden über externe Referenzen an die entsprechenden Objekte des Gebäudemodells gebunden (vgl. Abb. 3.3). Bei einer Änderung des Gebäudemodells in anderen Anwendungen wird eine neue IFC-Datei mit den Gebäudedaten generiert. Das hat zur Folge, dass die ursprünglichen Referenzierungen des Variationsmodells zum modifizierten Gebäudemodell verfallen. Ursache ist die Verwendung der IFC als Austauschformat und die damit verbundene Kurzlebigkeit des Modells. Um die Variation am veränderten Gebäudemodell dennoch vorzunehmen, muss ein neues Variationsmodell erstellt werden.

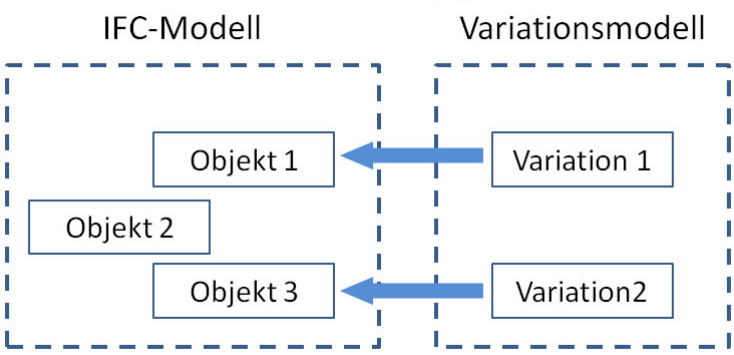


Abbildung 3.3: Variationsmodell mit externe Referenzierung

3.1.3 Relationsobjekte

Eine weitere Alternative wird durch die Verwendung von Relationsobjekten ermöglicht. Sie sind eigenständige Datenobjekte die Referenzierungen repräsentieren. Dateneinheiten lassen sich damit lose verbinden, da diese ihre Beziehung untereinander nicht explizit kennen müssen. Wie bei Assoziationen können auch Relationsobjekte zur direkten Erweiterung des IFC-Schemas verwendet werden. Im IFC-Datenmodell selbst wird ein Großteil der Referenzen durch Relationsobjekte umgesetzt (vgl. [Fuc15]). Die Klasse *IfcRelAssociates* erlaubt beispielsweise die Referenzierung zweier Objekte, die in einer Beziehung zueinander stehen. Desweiteren kann durch Relationsobjekte das Konzept des Multimodells angewendet werden. Multimodelle bündeln verschiedene Fachmodelle (auch Elementarmodelle) unterschiedlicher Domänen und bilden diese in einem ganzheitlichen Informationsraum ab (s. dazu auch [FN13, SS14]). Die Fachmodelle werden über sogenannte Linkmodelle lose miteinander verknüpft, wodurch die einzelnen Fachmodelle unberührt bleiben. Transformationsprozesse werden damit unnötig und etablierte Datenformate können weiterhin verwendet werden. Abbildung 3.4 veranschaulicht den strukturellen Aufbau eines Multimodells. Sie zeigt die Fachmodelle, 1 2 und 3, welche über die Linkmodelle 1 und 2 in Relation zueinander stehen. Der Austausch des Multimodells erfolgt über einen Multimodell-Container, welcher alle Komponenten(Link- und Fachmodelle) enthält.

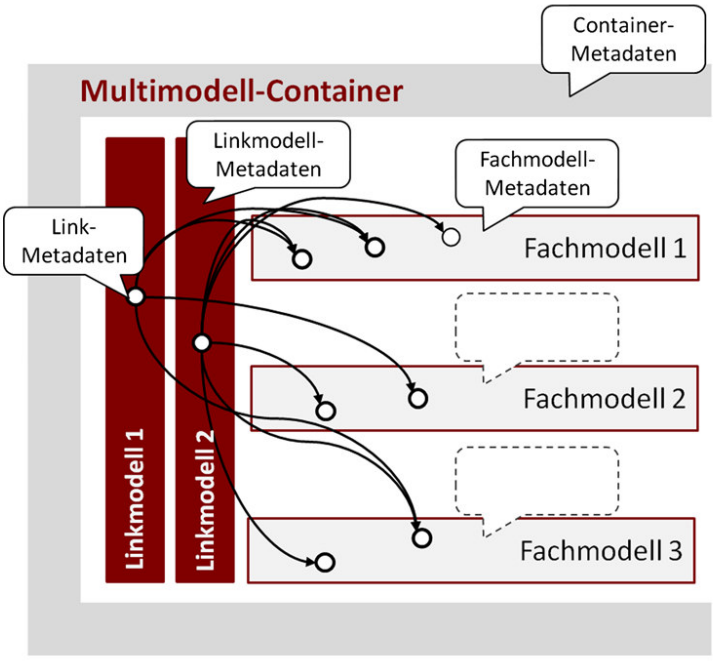


Abbildung 3.4: Struktureller Aufbau eines Multimodells [SS14, Abb. 2.7]

Der Ansatz des Multimodells findet im Bauwesen immer größeren Anklang. So entstand im Mefisto-Projekt[1] die universelle Multimodell-Software M2A2 sowie die Einführung der Multimodell-Fähigkeit in den Anwendungen iTWO von RIB und GRANID von GibGREINER.

[1] Mefisto ist ein Leitprojekt des Bundesministeriums für Bildung und Forschung (BMBF). Im Zeitraum von 2009 bis 2012 haben zwölf Partner aus Wissenschaft und Industrie in Mefisto neue Lösungen für das IT-gestützte Planen und Bauen erforscht.

Die Verknüpfung der Gebäudedaten mit den Parametervariationen kann auch mit der Anwendung des Multimodell-Ansatzes vorgenommen werden. Das Gebäudemodell und das Variationsmodell werden dabei über ein Linkmodell verknüpft (vgl. Abb. 3.5). Eine solche Konstellation bietet den Vorteil, dass beispielsweise vordefinierte Variationen für mehrere Modelle wiederverwendet werden können. Dazu müssen lediglich die Verknüpfungen der Relationsobjekte im Linkmodell bei einer Änderung der anderen Modelle aktualisiert bzw. neu gesetzt werden. Nachteilig gegenüber den anderen Varianten ist die Notwendigkeit eines zusätzlichen Modells (Linkmodell) sowie des Multimodell-Containers zum Austausch der Daten. Anzumerken ist zudem, dass der Ansatz des Multimodells verwendet wird, um multiple autonome Fachmodelle aus unterschiedlichen Domänen zu verknüpfen und damit domänenübergreifende Informationsräume zu bilden. Die Verwendung des Ansatzes für allein zwei Modelle ist daher unverhältnismäßig.

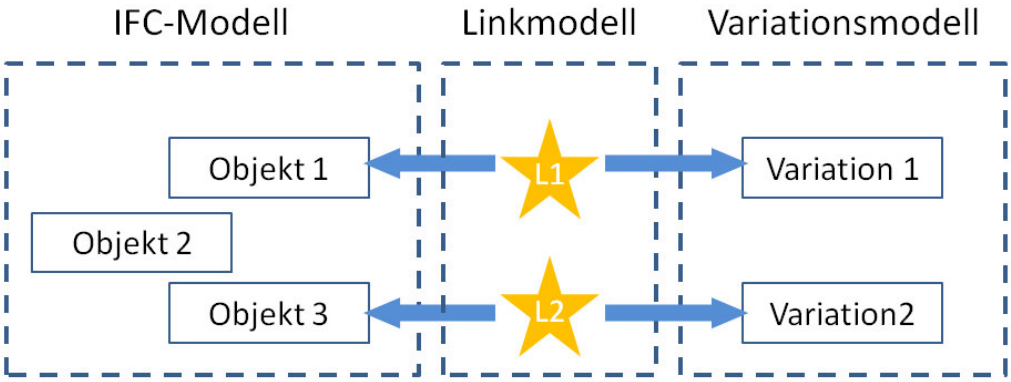


Abbildung 3.5: Variationsmodell mit Relationsobjekten

Ein weiterer Ansatz bildet die Property Set der IFC. Diese erlaubt die Integration von Daten, welche sich nicht durch das IFC-Schema beschreiben lassen. Sie werden im Folgenden genauer erläutert.

3.1.3.1 Property Set

Die Property Set ist eine besondere Funktionalität des IFC-Datenmodells. Sie bietet die Möglichkeit an, das IFC zu erweitern, ohne das Datenmodell ändern zu müssen. Dafür stellt die *Property Set* ein Metamodell innerhalb des IFC-Modells für die Beschreibung von benutzerdefinierten Eigenschaften zur Verfügung [Wix] [Lie09]. Die Möglichkeit das IFC-Datenmodell mit Property Sets zu erweitern, wurde vielfach wahrgenommen, so dass der Im- und Export sowie die Erstellung von Property Sets bei dem Großteil der IFC-fähigen Software möglich ist (z. B. ArchiCAD, Autodesk Revit, Nemetschek etc.). Nemetschek verwendet beispielsweise definierte *Property Sets* für Brandschutzeigenschaften von Bauteilen, wie z. B. ”PsetColumnCommon”, ”PsetSlabCommon” etc. [Pet07]. Ein weiteres Beispiel zeigt Abbildung 3.6. Die abgebildete Wand wurde mit Autodesk Revit erstellt und in IFC exportiert. Auf der rechten Seite der Abbildung sind die zugehörigen Property Sets der Wand aufgelistet (Abhängigkeiten, Sonstige etc.), welche automatisch von Autodesk Revit erstellt wurden. Sie werden beim Import des IFC-Gebäudemodells bei einem anderen Autodesk Revit-Anwender genutzt, um dieses in das softwarespezifische Gebäudemodell zu konvertieren.

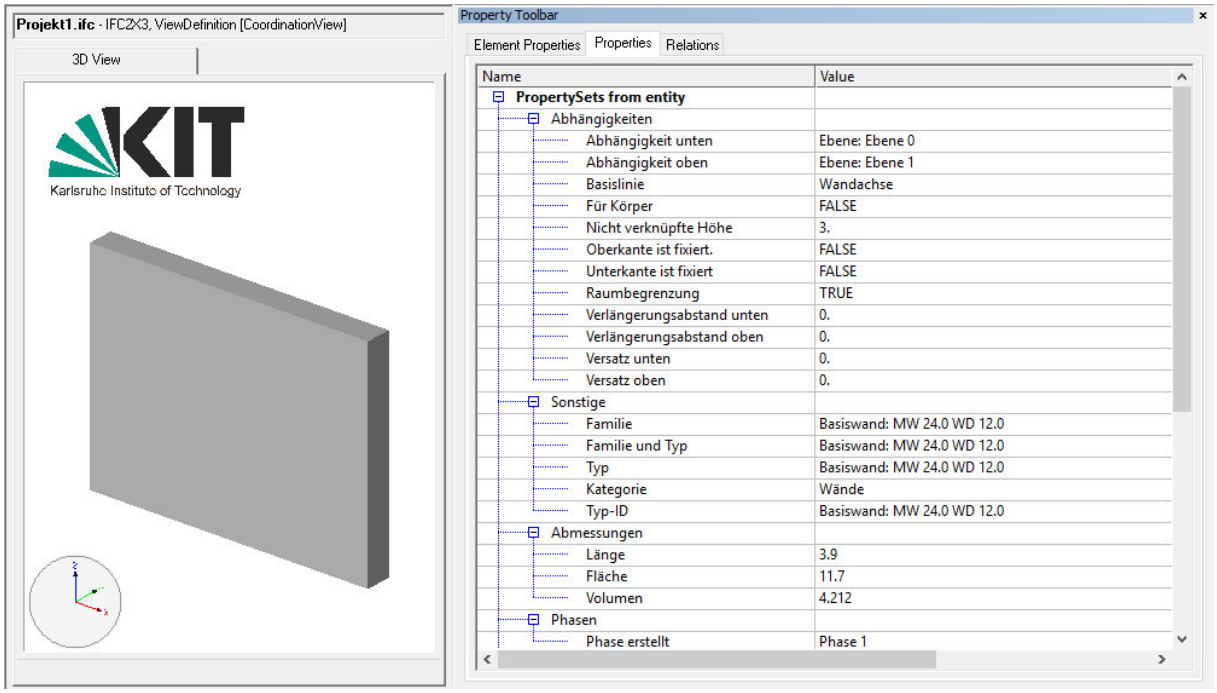


Abbildung 3.6: PropertySets einer in Autodesk Revit erstellten Wand

Im IFC-Datenmodell wird der Property Set-Mechanismus mithilfe der beiden Klassen *IfcPropertySet* und *IfcProperty* definiert und umgesetzt. Die darin definierten Eigenschaften werden über das Relationsobjekt *IfcRelDefinesByProperties* mit den Objekten des Gebäudemodells verknüpft (vgl. Abb. 3.7). Die *IfcPropertySet*-Klasse ist eine Container-Klasse und enthält die Eigenschaften (*IfcProperty*) in einer hierarchischen Baumstruktur. Sie fasst damit Eigenschaften in übergestellten Eigenschaftsgruppen zusammen. Die *IfcPropertys* werden dabei über die direkte Relation *HasProperties* mit dem jeweiligen *IfcPropertySet* in Beziehung gesetzt. Auf zwei Wegen können diese mit den zu spezifizierenden Objekten in Verbindung gebracht werden. Zum einen kann sie über die inverse Beziehung *DefinesType* zu einen Objekttypen (IfcTypeObject) gesetzt werden, wodurch alle zugehörigen Objekte des Objekttyps die entsprechenden Eigenschaften erhalten. Zum anderen können Property Sets auch direkt an einzelne Objekte über die inverse Beziehung *PropertyDefinitionOf* zugeordnet werden. Abbildung 3.7 (entnommen aus der IFC-Spezifikation) zeigt das Express-Schema der *IfcPropertySet*-Klasse.

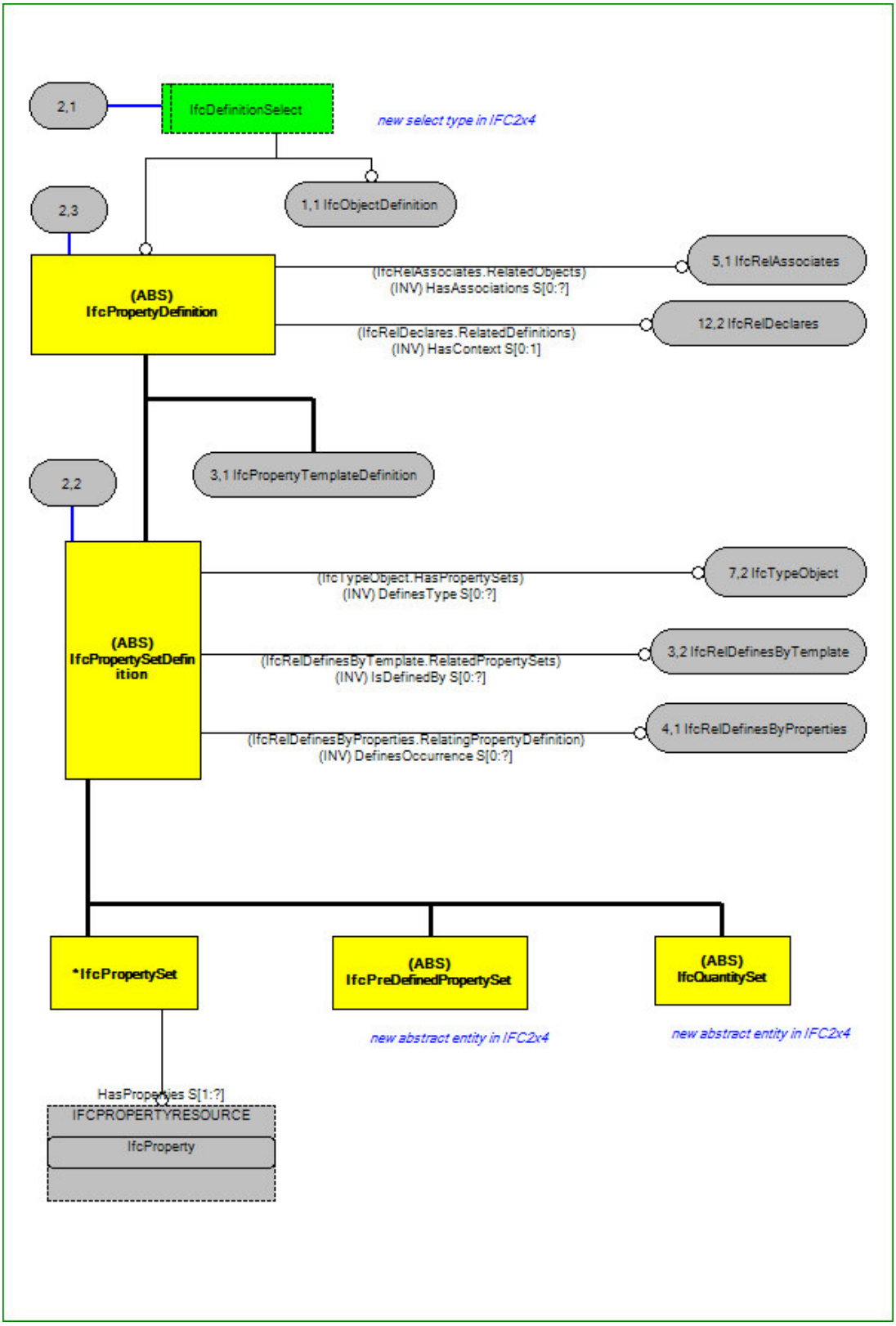


Abbildung 3.7: IfcPropertySet in EXPRESS-G [Bui]

IfcProperty repräsentiert die eigentlichen benutzerdefinierten Eigenschaften der Objekte. Sie werden über ihr Namens-Attribut (*Name*) eindeutig identifiziert und geben dem *IfcProperty* seine semantische Bedeutung innerhalb des Schemas. In Abbildung 3.8 ist die Struktur der Klasse *IfcProperty* aufgezeigt. Diese ist als abstrakt deklariert und es lassen sich deswegen keine direkten Objekte aus ihr bilden. Man unterscheidet jedoch insgesamt sieben verschiedene Sub- bzw. Subsubklassen die sich instanziieren lassen. Diese stellen die verschiedenen Arten zur Speicherung einer Eigenschaft dar und lassen sich grob in zwei Gruppen unterteilen:

- IfcSimpleProperty
- IfcComplexProperty

Die IfcSimpleProperty enthält die einfachen vom IFC-Schema vorgegebenen Formen der Speicherung von Eigenschaftswerten. Es können bei diesen immer fakultativ die Einheit (*IfcUnit*) der Eigenschaft mitgegeben werden. Man gliedert sie in sechs Unterklassen:

- **IfcPropertySingleValue**: speichert einen einfachen Wert vom Auswahltyp *IfcValue*[2].
- **IfcPropertyBoundedValue**: speichert zwei Werte vom Typ *IfcValue*. Diese repräsentieren eine obere und untere Schranke als Wertebereich der zu beschreibenden Eigenschaft.
- **IfcPropertyListValue**: speichert eine Liste von Elementen des Typs *IfcValue*.
- **IfcPropertyEnumeratedValue**: speichert einen oder mehrere Elemente aus einer vordefinierten Werteliste (Enumeration). Diese wird in einer *IfcPropertyEnumeration*-Instanz definiert und enthält Werte vom Typ *IfcValue*.
- **IfcPropertyTableValue**: speichert zwei gleichlange Listen aus Elementen vom Typ *IfcValue*. Diese stellen eine zweispaltige Tabelle dar, in denen Wertepaare abgelegt werden können.
- **IfcPropertyReferenceValue**: speichert eine Referenz zu einem anderen Objekt. Die Klasse des referenzierten Objektes muss dabei im Auswahltyp *IfcObjectReferenceSelect* enthalten sein.

Die *IfcComplexProperty* erlaubt die Definition von komplexeren Eigenschaften in einer Property Set. Dabei werden andere *IfcPropertys*, die als Liste in dieser enthalten sind, genutzt, um die Eigenschaft darzustellen.

Die Datenstruktur der Property Set erscheint damit als eine geeignete Lösung zur Implementierung von Parametervariationen in die IFC. Wie bei der direkten Erweiterung des IFC-Schemas verfallen die gesetzten Referenzierungen zwischen den Objektinstanzen des Gebäudemodells und den Daten des Variationsmodells nicht. Gleichzeitig entfällt die Hürde der Aufnahme des Variationsschemas in die IFC.

[2] IfcValue enthält benutzerdefinierte Typen für Maße, wie z. B. Winkel, Masse, Längeneinheiten etc., sowie allen IFC-spezifischen simplen Typen(s. Abschnitt 2.2.1.2).

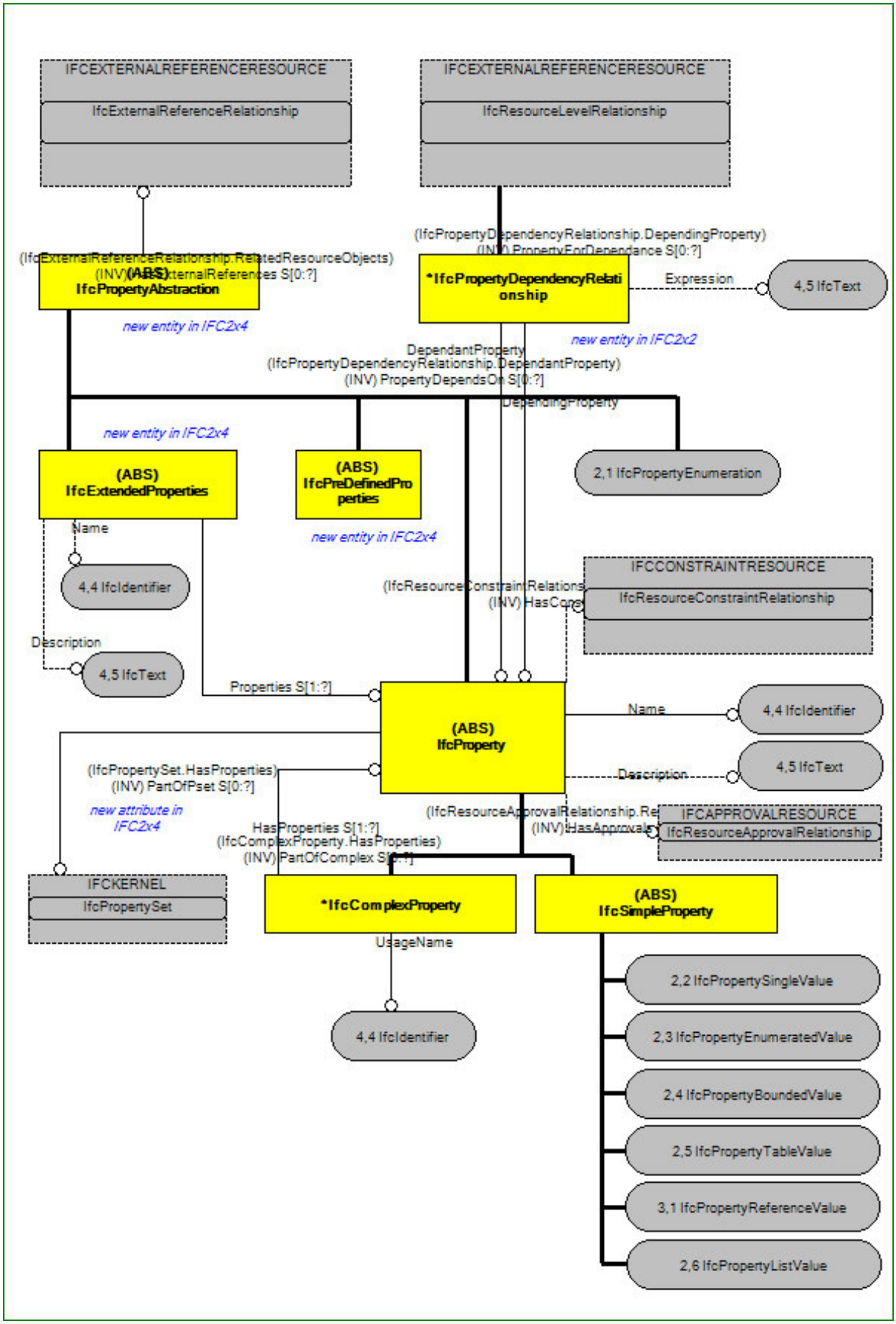


Abbildung 3.8: IfcProperty in EXPRESS-G [Bui]

3.2 Automatische Modellgenerierung

Parameterstudien sind in der Wirtschaft und Forschung unersetzlich. Bei den Versuchen werden eine Vielzahl von Parameterkonfigurationen ausgiebig getestet, um eine optimale Problemlösung zu finden oder Abhängigkeiten zwischen den Parametern festzustellen. So werden z. B. in der Medizin Studien zur Wirkung von Medikamenten gemacht, um die optimale chemische Zusammensetzung des Medikaments zu ermitteln. Auch im Bereich der Ingenieurwissenschaften werden Parameterstudien zur Lösung komplexer Probleme verwendet. Beispielsweise bei Materialuntersuchungen von Baustoffen oder Bemessungsaufgaben von Bauteilen. Mit dem Fortschritt der Technik werden diese auch immer öfter mit computergestützten Simulationen durchgeführt. Die für die Simulation benötigten Datenmodelle werden in der Regel mühsam per Hand erstellt, weshalb in die Modellerstellungen viel Zeit und Aufwand investiert werden muss. Dementsprechend sind die Bestrebungen für Verfahren zur automatischen Modellgenerierung groß.

IFC bieten dem Bauwesen ein neutrales Austauschformat für digitale Gebäudemodelle. Diese können von den einzelnen Gewerken mit ihren spezifischen Software-Anwendungen genutzt und bearbeitet werden. Als Austauschformat wird das Gebäudemodell in IFC jedoch nur für den Datenaustausch verwendet. Innerhalb der einzelnen Anwendungen werden aus der IFC-Datei interne Modellrepräsentationen zur weiteren Be- bzw. Verarbeitung gebildet. So wird z. B. in *Nemetschek* erst das Allplan Gebäudemodell erstellt und für den späteren Austausch in eine IFC-Datei exportiert [Pet07]. Durch diese Prämisse ist eine IFC-basierte vollautomatisierte Modellgeneration nur schwer möglich und wenig sinnvoll. Hingegen können Ansätze zur halb automatisierten Modellgeneration für IFC effektiv eingesetzt werden.

Im Brückenbau werden Bauteile meist durch parametrische Ausdrücke und mathematische Formeln beschrieben. Das STEP-Format erlaubt jedoch keine Variablen. Um das parametrische Verhalten in IFC dennoch nutzen zu können, wurde eine Schemaerweiterung von [NBY] vorgeschlagen. Mit dieser können Brückenbauteile semiautomatisch durch Formeln und Funktionen generiert werden. Die Erweiterung beschränkt sich jedoch nicht nur auf Brückenbauteile, sondern kann auf jegliche Objekte des IFC-Schemas angewendet werden. Der Ansatz der halb-automatischen Modellgenerierung wurde ebenfalls in [LGK14] für bauphysikalische Simulationen verwendet. Auf Basis eines IFC-Gebäudemodells wird dabei in drei Teilschritten ein Simulationsmodell zur bauphysikalischen Untersuchung generiert. Dazu werden im ersten Schritt die geometrischen Daten sowie thermische Materialeigenschaften aus dem Gebäudemodell gefiltert. Die in *<Solids>.xml* gespeicherten Geometriedaten werden im zweiten Schritt über den CBIP-Algorithmus weiter aufbereitet. Der Algorithmus ordnet den geometrischen Bauteilen des Modells einen der vier Elementtypen(*thermal elements, shading elements, opening elements* und *air boundary elements*) zu. Im letzten Schritt werden die thermischen Materialeigenschaften(*Materials.xml*) und das aufbereitete Geometriemodell(*SB.xml*) kombiniert und daraus das bauphysikalische Simulationsmodell generiert. Abbildung 3.9 veranschaulicht den Ablauf des dazu entwickelten Verfahrens. Ansätze der automatischen Modellgenerierung wurden also bereits erfolgreich für IFC-Modelle angewendet.

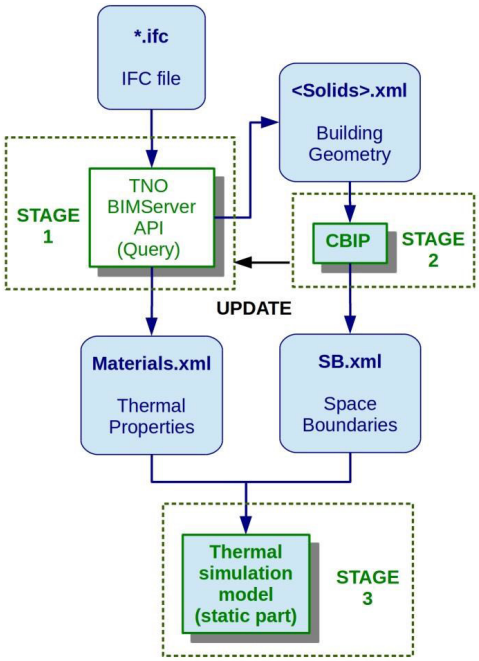


Abbildung 3.9: Modellgenerierungsprozess für thermische Berechnung aus IFC [LGK14, Fig 1.]

4 Entwicklung eines Konzepts zur Definition von Parametervariation

Im Rahmen dieser Arbeit wird ein Konzept für die Definition von Parametervariationen auf Basis von IFC-Daten entwickelt und verifiziert. Dazu wird zunächst ein Variationsmodell als Erweiterung des IFC-Modells erarbeitet und in Kapitel 5 mit der Entwicklung eines Interpreters und der Anwendung des Konzepts auf ein Beispiel getestet.

4.1 Variationsmodell

Der Begriff "Variationsmodell" wird in dieser Arbeit als eine Modellerweiterung verstanden. Diese soll eine halb automatisierte Generierung von Modellinstanzen auf Grundlage des ursprünglichen Gebäudemodells ermöglichen. Es werden dafür Variationen von Parameterbelegungen im Variationsmodell hinterlegt um die späteren Modellinstanzen zu erzeugen. Die Erweiterung kann dabei entweder in das Gebäudemodell integriert oder als externes Modell mit dem Ausgangsmodell verknüpft werden. Die Spezifikationen des Variationsmodells werden im Variationsschema formuliert.

4.1.1 Randbedingungen des Variationsmodells

Als ersten Schritt zur Entwicklung eines Konzepts für Parametervariation auf Basis von IFC-Gebäudemodellen wurden zuallererst Randbedingungen für die Konzeptlösung festgelegt. Diese sollen den Entwicklungsprozess des Variationsmodells in die IFC leiten und unterstützen. Es wurden für das Variationsmodell folgende Randbedingungen gestellt:

- Optionalität des Variationsmodells
- Atomare Variationen
- Komplexe Parametervariation
- Verwendung von EXPRESS und STEP

Im Folgenden wird erläutert weshalb die entsprechenden Randbedingungen für die Entwicklung des Variationsmodells gewählt worden sind.

4.1.1.1 Optionalität des Variationsmodell

Bei dem Variationsmodell handelt es sich um eine Erweiterung des bestehenden Gebäudemodells in IFC. Dieses soll als Hilfswerkzeug für die Speicherung von Variationen und deren Generierung

von weiteren Modellinstanzen dienen, ohne jedoch den Anwender in seiner Arbeit einzuschränken. Mit der Optionalität kann eine barrierefreie Erweiterung des IFC-Modells gewährleistet werden. So soll z. B. verhindert werden, dass bereits bestehende Gebäudemodelle durch die Einführung des Variationsmodells unbrauchbar werden und eine nachträgliche Implementierung von Parametervariationen möglich machen. Dazu ist es notwendig, dass das Variationsmodell auf die zu variierenden Modellobjekte zugreifen kann, ohne die ursprünglichen IFC-Modelldaten bzw. das Modellschema zu verändern. Die Parametervariationen des IFC-Objektes sollen deswegen über Referenzobjekte, die das zu variierende Objekt referenzieren, hinterlegt werden.

4.1.1.2 Variationen auf Attributebene

Die Parametervariationen, die mit dem Variationsmodell gespeichert werden, sollen atomar sein. Das heißt, dass die Parametervariationen auf Attributebene stattfinden sollen und nicht wie gewöhnlich auf Objektebene. Mit dieser Vorgehensweise vereinfacht sich die Struktur des Variationsmodells deutlich. Zudem reduziert sich der Speicheraufwand des Modells, da so nicht alle Attribute der Objekte im Variationsmodell abgebildet werden müssen. Änderungen von ganzen Objekten können mit multiplen Variationen auf Attributebene ausgeführt werden.

4.1.1.3 Komplexe Parametervariation

Für die Erstellung von vielseitigen Modellvariationen ist eine Möglichkeit zur Speicherung komplexer Parametervariationen notwendig. Die gespeicherten Variationen sollen sich nicht nur auf einfache Listen mit Wertebelegungen für einen Parameter beschränken. Stattdessen sollen Operationen, wie der Ausschluss eines Parameterwertes oder die Vereinigung zweier Wertemengen, für die Variation eines Parameters möglich sein. Ebenfalls sollen Parametervariationen im Variationsmodell kombiniert werden können. Durch die Kombination von variierten Attributwerten lassen sich z. B. Modelle mit mehr-tupligen Attributbelegungen schaffen, die für Studien notwendig sind. Dazu gehören Bemessungsprobleme bei der mehrere Parameter eines oder mehrere Bauteile untersucht werden. Die Ansätze für eine komplexe Parametervariation lassen sich damit grob in zwei Gruppen unterteilen:

- **Interne Variation**: Es werden verschiedene Operationen und/oder Strukturen zur Speicherung von Parameterwerten eines Attributes bereitgestellt.
- **Externe Interaktion**: Es werden Operationen für die Kombination zwischen mehreren variierten Parametern bereitgestellt. Anwendung findet diese Art der Variation beispielsweise bei der Bemessung eines Rundstützenquerschnitts, bei der unterschiedliche Kombinationen aus Radien und Festigkeiten für die Stütze geprüft werden sollen (s. Abb. 4.1).

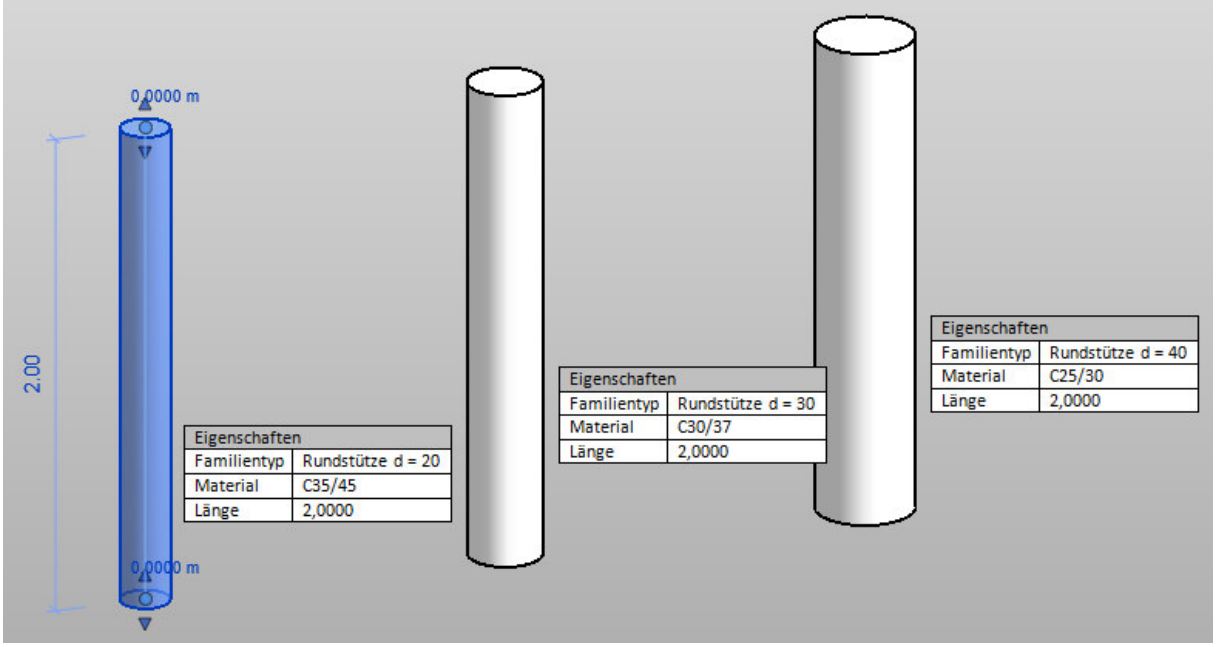


Abbildung 4.1: Kombinierte Variation von Durchmesser und Betonfestigkeit einer Rundstütze

Für beide Gruppen muss ein geeigneter Ansatz gefunden werden, um eine komplexe Parametervariation im Variationsmodell zu ermöglichen.

4.1.1.4 Verwendung von EXPRESS und STEP

Die Verwendung von EXPRESS für das Variationsmodell ist keine obligatorische Forderung. Das Modell kann auch in anderen Modellierungssprachen, wie z. B. Abstract Syntax Notation One (ASN.1), Standard Generalized Markup Language (SGML), EDIFACT, XSD etc.[1], formuliert werden. EXPRESS bietet sich, aufgrund einiger Vorteile gegenüber den anderen Datenmodellen, jedoch besonders an. Die Tatsache, dass das IFC-Modell auf EXPRESS basiert ist, bildet dabei der Hauptgrund dieser Bevorzugung. Bei einer späteren Implementierung ist kein weiterer Interpreter für das Variationsmodell notwendig, da für das Gebäudemodell und die Parametervariationen EXPRESS und STEP verwendet werden. Auch kann das Variationsmodell so leichter in das IFC-Modell integriert werden. Zwar gelten diese Vorteile ebenfalls für XSD und XML (s. Kapitel 2.1.2), jedoch wird von EXPRESS zusätzlich eine graphische Notation zur besseren Veranschaulichung bereitgestellt. Deswegen soll das Variationsmodell in EXPRESS formuliert werden und im STEP-Format ausgetauscht werden. Außerdem kann das Variationsmodell auch nachträglich, wie bei IFC, von EXPRESS und STEP zu XSD und XML konvertiert werden.

[1] genauere Ausführungen zu diesen Modellierungssprachen und ihren Datenmodellen können in [Sch93] gefunden werden

4.2 Variation mit Hilfe von Property Set

Zunächst soll eine Variationsdefinition mit Property Sets untersucht werden, da dieser Ansatz konform mit dem aktuellen IFC-Schema ist. Mit den Property Sets lassen sich viele unterschiedliche Daten, die das IFC-Datenmodell nicht abdeckt, integrieren ohne das IFC-Schema erweitern zu müssen. Für die Implementierung des Variationsmodells in die IFC soll deshalb zunächst das gegebene Property Set-Konzept verwendet werden. Dafür werden neue Property Sets eingeführt, welche die für das Variationsmodell erforderlichen Daten als Eigenschaften beinhalten sollen. Über die Relation *IfcRelDefinesByProperties* werden die Property Sets mit dem zu variierenden Objekt verknüpft (s. Abb. 4.2). Durch diese Art der Verknüpfung muss das ursprüngliche Objekt nicht verändert werden. Die Forderung nach der Optionalität des Variationsmodells ist damit erfüllt. Hierbei ist zu bemerken, dass die Referenzierung bisher nur auf das zu variierende Objekt gesetzt wurde. Eine direkte Beziehung zwischen der Variation des Attributs und dem entsprechenden Attribut selbst kann in IFC nicht umgesetzt werden, da Relationen nur zwischen Entitäten bestehen können. Neben den Variationswerten soll deshalb auch ein Verweis auf das spezifische Attribut in den Property Sets hinterlegt werden.

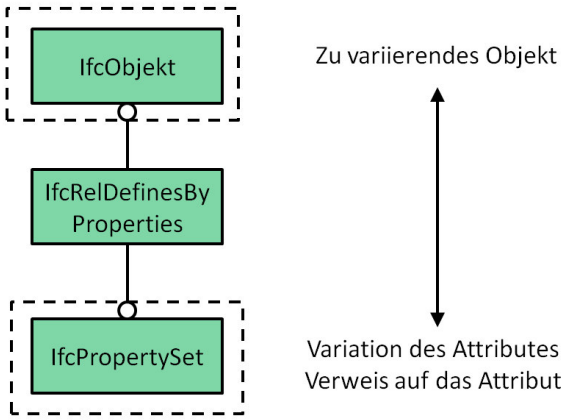


Abbildung 4.2: Verknüpfung der Property Set mit dem Objekt

Die erste Eigenschaft des Property Sets soll den Verweis auf das Attribut repräsentieren. Es soll dafür ein String mit dem Namen des zu variierenden Attributs gespeichert werden. Die semantische Bedeutung des Strings wird mithilfe des IFC-Schemas ermittelt. Dadurch entsteht keine reelle Referenz zum eigentlichen Attribut, jedoch kann dieses nun eindeutig innerhalb der Entität identifiziert werden. Als Alternative ist auch die Verwendung von ganzen Zahlen, welche die Position des Attributs in der Instanzdeklaration angibt, möglich. Diese Möglichkeit wurde jedoch verworfen, da die Methodik nur IFC-intern funktioniert. Da der Großteil der Anwendungen mit ihren spezifischen Modellen arbeiten und diese erst für den Austausch in IFC konvertieren, können sie nicht angewendet werden.

Weiterhin wird eine Property zur Speicherung der Parametervariation für die Implementierung eines Variationsmodells über die Property Set-Struktur benötigt. Dabei ist zu bedenken, dass verschiedene Möglichkeiten für die Art der Speicherung existieren. Eine Auswahl wird im Fol-

genden aufgeführt:

- **Werteliste**: Einzelne Variationswerte werden in einer geordneten oder ungeordneten Liste direkt aufgeführt. Die Werteliste bietet sich besonders für beliebige Wertemengen an.
- **Wertebereich**[2]: Es werden die untere und obere Schranke eines Wertebereichs hinterlegt. Zusätzlich muss eine Schrittweite angegeben werden. Die einzelnen Elemente stehen in einer linearen Beziehung zueinander.
- **Formeln zur Berechnung der Werte**[2]: Diese Art der Speicherung ist eine erweiterte Form des Wertebereichs, bei der die Mengenelemente auch nicht-lineare Beziehung zueinander besitzen können. Diese Beziehung wird mit einer mathematischen Funktion beschrieben und ist daher auf numerische Werte beschränkt.

Beispielhaft sollen mögliche Ausprägungen von Properties für die Speicherung mit Wertelisten und Wertebereichen aufgezeigt werden. Die Werteliste lässt sich mit der Property vom Typ *IfcPropertyListValue* leicht implementieren, da die gegebene Struktur zur Speicherung von Listenelementen geschaffen wurde. Auch für die Implementierung von Wertebereichen bietet IFC, mit *IfcPropertyBoundedValue*, eine geeignete Klasse an. Es ist dabei jedoch keine Struktur für die Schrittweite gegeben. Damit diese modelliert werden kann, muss eine weitere Property vom Typ *IfcPropertySingleValue* eingeführt werden um die Schrittweite zu speichern. Aus den vorangegangenen Beschreibungen können die zwei Property Sets *Pset_ListedVariance* und *Pset_ListedVariance* erstellt werden. Tabelle 4.1 und 4.2 zeigen diese beiden Property Sets und deren enthaltenen Properties.

Tabelle 4.1: Properties von Pset_ListedVariance

Property Name	Property Type	Datentyp	Beschreibung
AttributeBinding	SingleValue	IfcLabel	Name des zu variierenden Attributs
Variance_List	ListValue	IfcString	Liste aus den Variationswerten

Tabelle 4.2: Properties von Pset_BoundedVariance

Property Name	Property Type	Datentyp	Beschreibung
AttributeBinding	SingleValue	IfcLabel	Name des zu variierenden Attributs
Variance_Bound	BoundedValue	IfcValue	Wertebereich der Variationswerte
Bound_Increment	SingleValue	IfcValue	Schrittweite des Wertebereichs

[2]Diese Varianten der Speicherung sind nur für numerische Werte (Datentyp NUMBER) möglich. Zeichen bzw. Zeichenfolgen können nicht verwendet werden da sie keiner logischen Ordnung folgen, die auch ihrer semantischen Bedeutung entspricht.

Mit diesen Property Sets können Parametervariationen innerhalb des IFC-Modells auf unterschiedliche Art gespeichert werden. Die Variationen können als Werteliste oder als Wertebereich (mit Schrittweite) hinterlegt werden. Dabei werden einzelne Attribute und nicht ganze Objekte variiert. Durch die Nutzung der Property Set-Mechanik ist somit die Optionalität des Variationsmodells gewahrt. Ebenso wurde die Forderung nach der Verwendung von EXPRESS und STEP nachgekommen, da das Modell auf dem IFC-Datenmodell basiert[3] und damit auch auf EXPRESS. Die in 4.1.1 beschriebenen Randbedingungen sind dennoch nicht vollständig erfüllt. Zwar wurden zwei Arten der Speicherung bereitgestellt, weswegen trotzdem noch nicht von einer komplexen Parametervariation gesprochen werden kann. So sind z. B. der Ausschluss von einzelnen Werten oder die Kombination von ganzen Variationen mit dem aktuellen Stand nicht ausführbar. Zusätzlich grenzt die Struktur der Property Sets die Art, wie das Modell aufgebaut werden kann, stark ein und erschwert die Implementierung einer solchen Modifizierung. Das größte Defizit ist jedoch die Auswahl des zu variierenden Objekts. Mit den Property Sets ist der Verweis nur auf Instanzen der Klasse *IfcObject* beschränkt. Diese beinhaltet zwar alle physikalischen Gegenstände, konzeptionelle Gegenstände, Prozesse und Akteure, aber die zu variierenden Attribute, wie z. B. Geometrieeigenschaften, befinden sich größtenteils in anderen IFC-Klassen. Die Breite eines Trägers ist beispielsweise nicht in der Klasse *IfcBeam* hinterlegt, sondern über verschiedene Relationen als Attribut in einer Instanz von *IfcRectangleProfileDef* enthalten (s. Abb. 4.3). Noch größer wird die Problematik mit der Universalität der IFC. Die Möglichkeit, einen Träger auf verschiedene Arten darzustellen, ist für gewöhnlich eine Stärke der IFC, jedoch werden bei unterschiedlichen Darstellungsarten auch unterschiedliche Klasseninstanzen verwendet. Das Mapping vom Objekt zum konkreten Attribut stellt damit eine große Schwierigkeit dar.

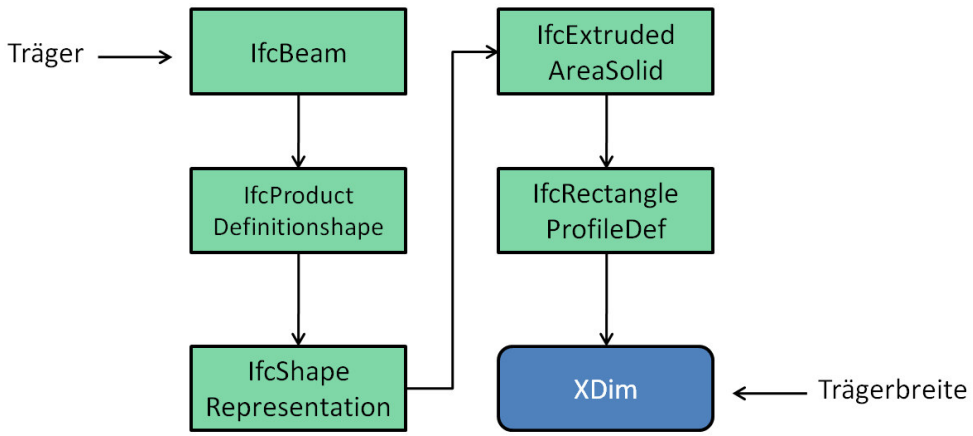


Abbildung 4.3: Pfad von der Trägerklasse zum Attribut der Trägerbreite

Zusammenfassend kann der Ansatz des Variationsmodells über die Property Set-Mechanik als unzureichend betrachtet werden. Die Bindung an die von IFC vorgegebene Property Set-Struktur erschwert das Finden einer geeigneten Lösung der vorher genannten Probleme. Der Ansatz des Variationsmodells auf Basis der Property Sets wurde deshalb verworfen.

[3]Es wird hier von einer Verwendung des IFC EXPRESS ausgegangen und nicht von IFC XML.

4.3 IFC-Erweiterung

Aus den vorangegangenen Versuchen, Parametervariationen über Property Sets zu implementieren, konnten Problempunkte erkannt werden. Die Hauptproblematik bei der Einführung des Variationsmodells ist neben der Integration der Schemaerweiterung die Anbindung der Variationen an das Objekt bzw. das Attribut und die Art und Struktur der Variation selbst. Im Abschnitt 3.1 wurden bereits Formen der Schemaerweiterung mithilfe von verschiedenen Linksystemen untersucht. Die Untersuchungen haben gezeigt, dass die übrigen betrachteten Linksysteme keine optimale Lösung für die Implementierung von Parametervariationen bieten. Tabelle 4.3 zeigt die Vor- und Nachteile der verschiedenen Linksysteme zur Integration des Variationsmodells auf. Für die Umsetzung des Modells wird daher eine Mischform zwischen externen Referenzen und Relationsobjekten verwendet. Wie beim Ansatz der externen Referenz befinden sich die Parameterwerte der Variationen und die Referenzierungen zu den Objekten des Gebäudemodells im Variationsmodell. Sie werden voneinander entkoppelt und über ein Relationsobjekt in Relation gesetzt (vgl. Abb. 4.4). Dadurch entsteht ein Platzhalter für die externe Referenzierung einer Parametervariation, ähnlich dem Ansatz des Multimodells. Bei Änderungen im Gebäudemodell muss das Variationsmodell dennoch angepasst werden. Die Variationswerte bleiben dabei jedoch unberührt und es werden allein die Referenzierung zum Gebäudemodell modifiziert. Dadurch werden die Vorteile beider Ansätze, ohne die Notwendigkeit eines Multimodell-Containers oder Linkmodells, vereint. Dafür werden im Folgenden geeignete Teilmodelle für die Variationswerte und deren Referenzierung bzw. Zeiger gebildet. Anschließend sollen diese Teilmodelle zu einem ganzheitlichen Variationsmodell zusammengefügt werden. Für die Modellierung soll, wie in den Randbedingungen gefordert, EXPRESS verwendet werden.

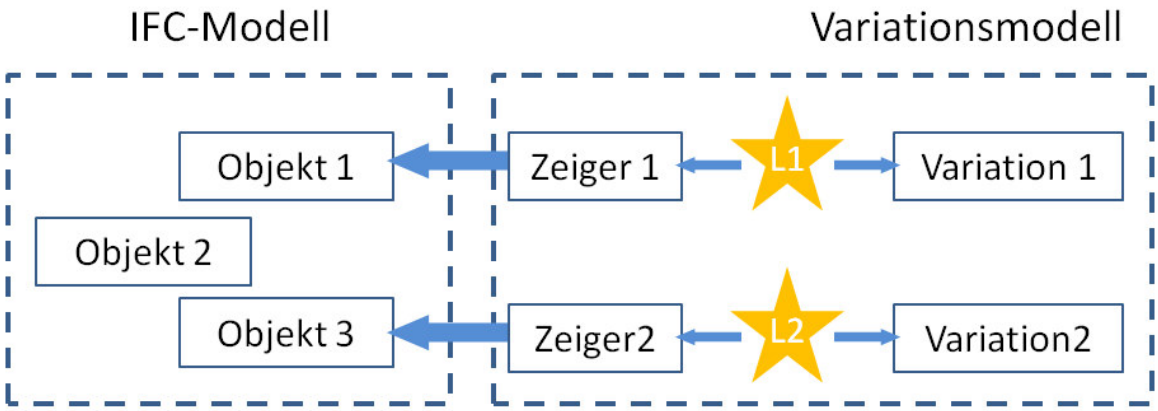


Abbildung 4.4: Variationsmodell mit Ansatz aus externer Referenzierung und Relationsobjekten

Tabelle 4.3: Vor- und Nachteile der Linksysteme zur Integration des Variationsmodells

	Vorteile	Nachteile
Direkte Schemaerweiterung	• Referenzierungen des Variationsmodells verfallen nicht bei Änderung des Gebäudemodells • keine zusätzlichen Modelle notwendig	• Aufnahme in das IFC-Schema als Teilmodell notwendig • Für die Generierung der Modellinstanzen sind Schnittstellen in den Software-Anwendungen erforderlich
Externe Referenzierung	• IFC-Schema muss nicht verändert werden	• Parametervariationen verfallen bei Änderung des Gebäudemodells
Multimodell	• IFC-Schema muss nicht verändert werden • erstellte Variationen im Variationsmodell bleiben erhalten und können für andere Gebäudemodelle wiederverwendet werden	• Multimodell-Container und Linkmodell/-e sind neben dem Variationsmodell erforderlich • Referenzierungen verfallen bei Änderungen des Gebäude- bzw. Variationsmodell
Property Sets	• IFC-Schema muss nicht verändert werden • es ist nur ein Metamodell für die Property Sets erforderlich • Schnittstellen für Property Sets sind größtenteils vorhanden	• Gebunden an die Struktur der Property Sets

4.3.1 Modell zur Verknüpfung der Variation an Attribute

Um Eigenschaften von Objekten zu variieren, muss eine Referenz auf das jeweilige Attribut des Objekts gesetzt werden. Der Zugriff muss dabei über das Objekt erfolgen, da Relationen nur zwischen Objekten möglich sind. Wie im Ansatz der Property Sets (s. Kapitel 4.2) wird ebenfalls ein Zeiger auf das zu variierende Attribut gesetzt. Dazu soll der Attributname als eindeutiges Unterscheidungsmerkmal dienen. Die bisher beschriebenen Eigenschaften der zu bildenden Referenz-Klasse waren bereits im Ansatz der Property Sets enthalten (s. Kapitel 4.1.1). Es wurden bisher nur Attribute mit einer 1:1-Kardinalität berücksichtigt. Es sind allerdings auch Attribute mit höheren Kardinalitäten in EXPRESS möglich, wie z.B. bei *IfcCartesianPoint*. Diese Klasse repräsentiert einen Punkt in einem 2- oder 3-dimensionalen Raum und besitzt das Attribut *Coordinates*, welches die Punktkoordinaten als Liste enthält. Für solche Aggre-

gationstypen ist ein weiteres Attribut der Referenz-Klasse notwendig, um die Forderung nach atomaren Variationen zu erfüllen (vgl. Kapitel 4.1.1.2). Ein einfacher numerischer Wert reicht hierbei für die Identifikation aus. Abbildung 4.5 zeigt die Referenzklasse mit den vorangegangen Spezifikationen.

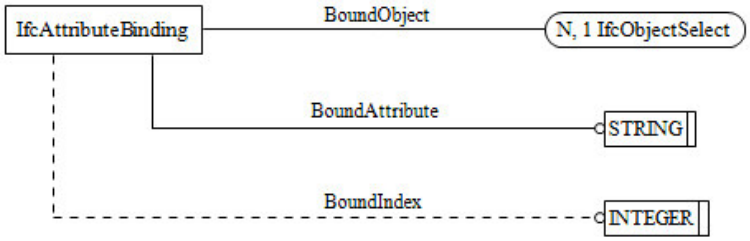


Abbildung 4.5: IfcAttributeBinding in EXPRESS-G

Mit der Klasse *IfcAttributeBinding* lassen sich alle Attribute der in *IfcObjectSelect* enthaltenen Klassen referenzieren. Der Auswahltyp *IfcObjectSelect* muss daher nahezu alle geometrisch und statisch relevanten Klassen umfassen. Eine erste Vorstellung, welche Entitäten das betrifft, können aus dem *IfcParametricObjectSelect* des Schemas IfcParametric2x3 aus [NBY] entnommen werden, an welches das Modell zur Verknüpfung von Variation angelehnt ist. Die *IfcParametricObjectSelect* wird ebenfalls zur Referenzierung von Objekte verwendet. Jedoch werden diese dazu genutzt, parametrische Beziehungen aufzubauen. Sie enthält aus diesem Grund nur geometrisch- und materialspezifische Klassen. Eine Erweiterung für die *IfcObjectSelect* wird an dieser Stelle nicht vorgenommen. Die Tabelle 4.4 listet die in *IfcObjectSelect* enthaltenen Klassen auf.

Tabelle 4.4: Klassen des Auswahltyps IfcObjectSelect

IfcPropertySingleValue	IfcBoundingBox	IfcProject
IfcPropertyEnumerationValue	IfcQuantityArea	IfcSlab
IfcThermalMaterialProperties	IfcBuilding	IfcDoor
IfcRectangleProfileDef	IfcWindowStyle	IfcSpace
IfcExtrudedAreaSolid	IfcWallType	IfcRoof
IfcQuantityVolume	IfcWindow	IfcSlabType
IfcCartesianPoint	IfcDirection	IfcSite
IfcMaterialLayer	IfcDoorStyle	IfcWall
IfcBuildingStorey	IfcQuantityLength	

Als Beispiel soll hier die Referenzierung der y-Koordinate eines kartesischen Punktes aufgeführt werden (s. Listing 4.1). Mit dem *BoundObject*-Attribut wird auf den spezifischen Punkt (*IfcCartesianPoint*) verwiesen. Das *BoundAttribute* identifiziert das betreffende Attribut über dessen Attributnamen, in diesem Fall dem Attribut *Coordinates*. Es muss zusätzlich ein Index im *BoundIndex* hinterlegt werden, da sich bei *Coordinates* um einen Aggregationstyp handelt. Für die Referenzierung der y-Koordinate wird der Wert 1 übernommen[4].

[4]Zähler beginnen in der Informatik üblicherweise mit 0, weswegen die x-,y- und z-Koordinate mit 0,1 und 2

Listing 4.1: Beispiel Referenzierung einer y-Koordinate

```
1 DATA;
2 #8=IFCCARTESIANPOINT((0.,1.,2.));
3 #9=IFCATTRIBUTEBINDING(#8,'COORDINATES',1);
4 ENDSEC;
```

Die *IfcAttributeBinding* ermöglicht somit eine umfangreichere Referenzierung von Attributen, als sie im Ansatz der Property Sets realisierbar war. Die Auswahl der Objekte beschränkt sich nicht mehr nur auf die Klasse *IfcObject*, sondern ist abhängig von der in *IfcObjectSelect* enthaltenen Klassen. So kann die Auswahlmöglichkeit auf alle IFC-Klassen erweitert werden. Generell sollte der Umfang von *IfcObjectSelect* auf das nötigste beschränkt werden.

4.3.2 Modell zur Variation der Werte

Im vorangegangen Ansatz der Property Set (s. Abschnitt 4.2) wurden verschiedene Strukturen zur Speicherung von Variationen diskutiert und verwendet. Operationen zwischen den Variationen, wie den Ausschluss einzelner Variationswerte, waren durch die vorgegebene Struktur der Property Sets jedoch nur schwer umsetzbar. Deswegen soll im Modell zur Variation der Werte ein stärkerer Fokus auf Kombinationen von Variationswerten gelegt werden. Als Speicherstruktur für die Variationswerte wurden Listen gewählt, da Listen universell anwendbar sind und andere Speicherstrukturen wie Wertebereiche und Formeln in ihr abgebildet werden können. Abbildungen, z. B. von Wertebereichen, auf Listen haben teilweise einen höheren Speicherverbrauch zur Folge, da sie unkomprimiert abgelegt werden. Dieses Defizit wird für die Universalität der Listen in Kauf genommen. Über einen Editor für das Variationsmodell können auch mit der bloßen Verwendung der Listenstruktur verschiedene Eingabemöglichkeiten bereitgestellt werden (z. B. Wertebereich und Formeln). So muss der Anwender beispielsweise für die Variation eines Wertes von 1 bis 100 nicht alle Werte einzeln eingeben sondern kann alternativ den Wertebereich 1 bis 100 mit Schrittweite 1 angeben. Die Eingabe sollte anschließend automatisch in die Listenstruktur umgeformt werden.

Für die Speicherung der Variationswerte wird die Klasse *IfcAttributeVariance* eingeführt. Listing 4.2 zeigt die Deklaration der Klasse. Die Werte des zu variierenden Parameters werden im Attribut *VarianceList* hinterlegt und sollen vom Auswahltyp *IfcValue* sein. Zusätzlich wird ein Default-Wert im Attribut *Default* abgelegt, der für die Kombination von mehreren Parametervariationen notwendig wird.

Listing 4.2: IfcAttributeVariance in EXPRESS

```
1 ENTITY  IfcAttributeVariance;
2 Default:         IfcValue;
3 VarianceList:    LIST [0:?] OF IfcValue;
4 END_ENTITY;
```

angesprochen werden.

Um Variationswerte aus einer anderen Menge auszuschließen oder mehrere Mengen zu vereinen ist die Einführung von Mengenoperationen innerhalb des Variationsmodells erforderlich. Eine ähnliche Problematik wurde bereits für CSG-Körper in IFC gelöst. Bei der Constructive Solid Geometry (CSG) werden komplexe 3D-Körper und Oberflächen mithilfe von booleschen Operatoren zwischen einfachen geometrischen Körpern, wie z. B. Würfel, Kugel, Zylinder etc., gebildet. In IFC wird das Ergebnis einer solchen booleschen Operation von der Klasse *IfcBooleanResult* repräsentiert. Diese enthält in ihren Attributen die zwei Körper, an denen die boolesche Operation vorgenommen wird (*FirstOperand* und *SecondOperand*), und die Art der booleschen Operation(*Operator*). Dabei ist besonders anzumerken, dass der *FirstOperand* und der *SecondOperand* vom Datentyp *IfcBooleanOperand* sind. Dieser Datentyp erlaubt die Auswahl zwischen anderen geometrischen Körpern (*IfcSolidModel*, *IfcHalfspaceSolid* und *IfcCsgPrimitive3D*) sowie anderen Operationsprodukten (*IfcBooleanResult*). Das hat zur Folge, dass der boolesche Produktkörper ineinander verschachtelt werden kann. Diese Struktur soll auch für die Parametervariationen nachgebildet werden. Dazu wird der Datentyp *IfcVarianceOperand* und die Klasse *IfcVarianceResult* eingeführt. Ersteres ist ein Auswahltyp und bietet die Wahl zwischen *IfcAttributeVariance* und *IfcVarianceResult*. Letzteres ist die Klasse, welche das Ergebnis einer booleschen Operation zwischen zwei Mengen bildet. Listing 4.3 zeigt die Deklarationen der beiden Typen in EXPRESS.

Listing 4.3: IfcVarianceOperand und IfcVarianceResult in EXPRESS

```
1  TYPE IfcVarianceOperand = SELECT
2  (IfcVarianceResult,
3  IfcAttributeVariance);
4  END_TYPE;
5
6  ENTITY  IfcVarianceResult;
7  FirstOperand:   IfcVarianceOperand;
8  SecondOperand:  IfcVarianceOperand;
9  BooleanOperator:IfcBooleanOperator;
10 END_ENTITY;
```

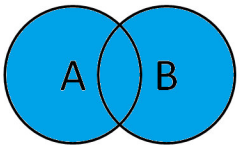
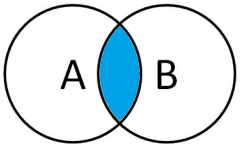
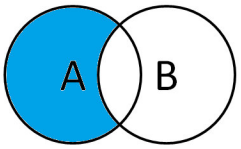
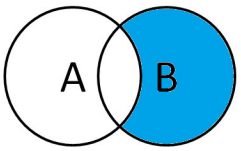
Mit den Mengenoperationen lassen sich aus verschiedenen Mengen neue Mengen bilden. Es wird zwischen den folgenden Operationen unterschieden:

- Vereinigung / Adjunktion
- Durchschnitt / Konjunktion
- Differenz
- Symmetrische Differenz

Die Mengenoperationen, die für die Parametervariationen angeboten werden, sollen den des Datentyps *IfcBooleanOperator* entsprechen. Darin entfällt von den vier genannten Operationen die Symmetrische Differenz, welche als Operation für Parameter ohnehin schon wenig sinnvoll ist.

Tabelle 4.5 zeigt die übrigen in *IfcBooleanOperator* angebotenen Operationen. Die Vereinigung und Durchschnitt sind kommutative Operationen[5]. Dagegen ist die Differenz nicht kommutativ (d.h. A\B ist ungleich B\A in nichttrivialen Fällen). Der Enumerationstyp *IfcBooleanOperator* wird aus den IFC in das Variationsmodell importiert und gibt den Operator für die Verknüpfung zweier Variationslisten an.

Tabelle 4.5: Mögliche Operationen von IfcBooleanOperator

Enumeration	**Operation**	**Symbol**	**Beispiel**
.UNION.	Vereinigung	A ⋃ B	
.INTERSECTION.	Durchschnitt	A ⋂ B	
.DIFFERENCE.	Differenz	A \ B	
		B \ A	

Parameterwerte werden im Variationsmodell als Listenelemente hinterlegt. Die einfache Verwendung von Mengenoperation mit Listenelementen ist daher problematisch. Im Gegensatz zu Listen gibt es in Mengen keine Ordnung zwischen den Elementen. Ebenso dürfen Werte nicht mehrfach auftreten. Deswegen wird eine Konvention für die Anwendung der Mengenoperation getroffen. So wird die Werteliste für die Umsetzung der Operation in eine Wertemenge umgewandelt, d.h. es liegt keine Reihenfolge mehr vor und mehrmalig auftretende Werte werden bis auf ein Exemplar gekürzt. Mit dieser Konvention lässt sich die Spannung zwischen Listenwerte und Mengenwerte der Variation lösen. Das Ergebnis aus der Operation wird anschließend aufsteigend sortiert und somit in eine Liste umgeformt. Die vorangegangenen Wertedopplungen werden nicht berücksichtigt. Ein Beispiel für die Bearbeitung einer Parametervariation aus einer Mengenoperation zeigt Abbildung 4.6.

[5] Argumente dieser Operation können vertauscht werden, ohne das Ergebnis zu ändern.

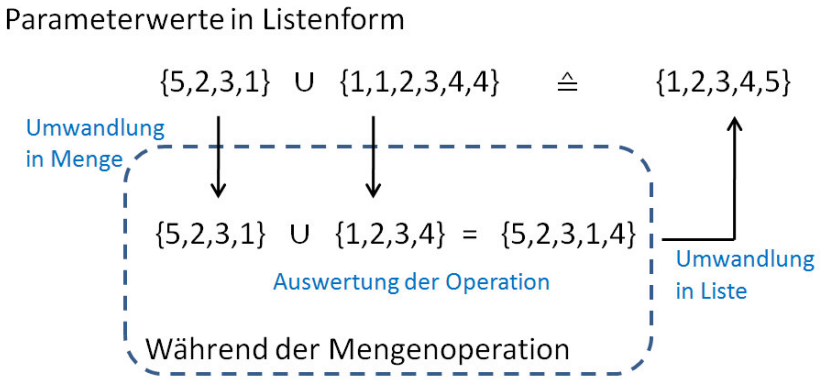


Abbildung 4.6: Händeln derVariationswerte während einer Mengenoperation

Mit den soeben beschriebenen Datentypen lassen sich Variationswerte in einer Listenstruktur abspeichern. Zudem können an den Variationen Mengenoperationen durchgeführt werden. Dem Anwender ist damit eine reiche Vielfalt an Möglichkeiten gegeben, die Variationswerte eines Parameters zu speichern. Das komplette Modell zur Speicherung von Variationswerten wird in Abbildung 4.7 in EXPRESS-G aufgeführt. Die Forderung nach einer komplexen Parametervariation (s. Abschnitt 4.1.1.3) konnte bis hierhin nur teilweise erfüllt werden, da die Bedingung für externe Interaktionen zwischen mehreren Parametern noch nicht erfüllt wurde. Dieser Forderung soll im Folgenden Abschnitt nachgegangen werden.

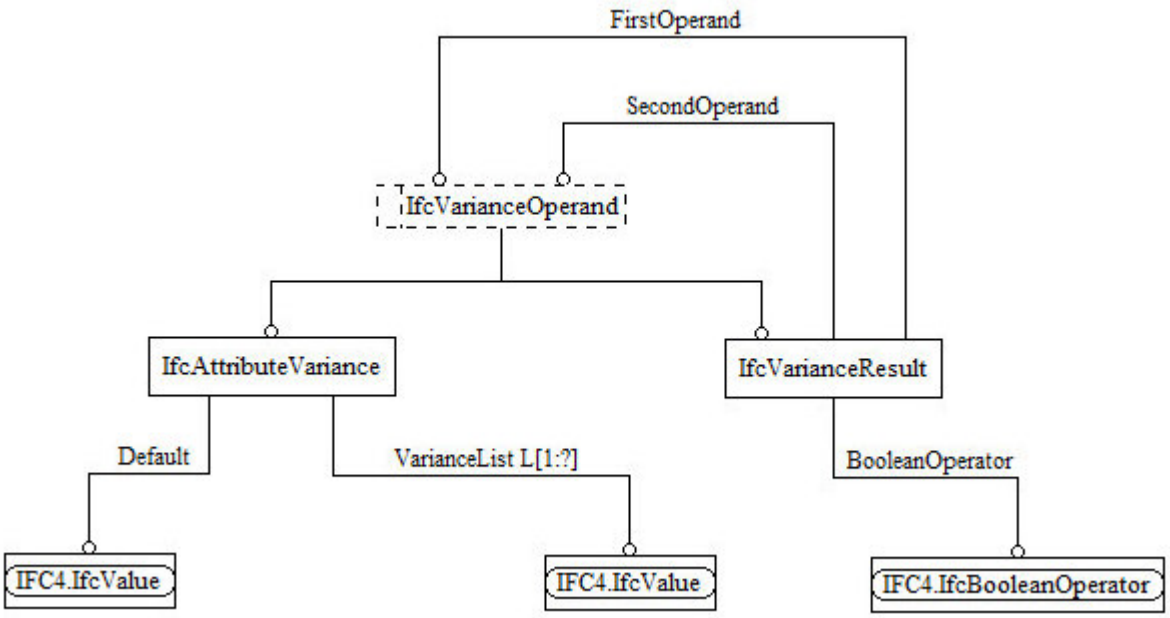


Abbildung 4.7: IfcAttributeVariance in EXPRESS-G

4.3.3 Zusammenführung der Teilmodelle

Für das Variationsmodell wurden in den vorangegangen Abschnitten 4.3.1 und 4.3.2 Teilmodelle für die Verknüpfung der Parametervariationen und deren Speicherung modelliert. In diesem Abschnitt sollen diese nun zu einem vollständigen Variationsmodell zusammengefügt werden. Dazu wird der neue Datentyp *IfcVarianceConnector* erstellt (s. Listing 4.4). Dieser soll die Attribute *BoundVariance* und *BoundAttribute* besitzen. Die *BoundVariance* ist vom Datentyp *IfcVarianceOperand* und enthält die Variationswerte des zu variierenden Parameters, welche entweder eine einfache Liste (*IfcAttributeVariance*) oder das Ergebnis aus einer einfachen oder mehreren geschachtelten Mengenoperationen (IfcVarianceResult) von Wertelisten sind. Das Attribut *BoundAttribute* ist vom Datentyp *IfcAttributeBinding*, welches den Zeiger auf den Variationsparameter umfasst. Durch die beiden Attribute werden zwei Relationen mit einer 1:1-Kardinalität aufgebaut und so die Variationsparameter mit den dazugehörigen Variationswerten verknüpft.

Listing 4.4: IfcVarianceConnector in EXPRESS

```
1 ENTITY  IfcVarianceConnector;
2 BoundVariance   :       IfcVarianceOperand;
3 BoundAttribute  :       IfcAttributeBinding;
4 END_ENTITY;
```

Mit dem bisherigen Stand lassen sich bereits einzelne Parametervariationen mit dem Variationsmodell abspeichern. Abbildung 4.8 zeigt schematisch eine beispielhafte Variation einer Stützenbreite mit dem jetzigen Ansatz des Variationsmodells im STEP-Format. Die Breite der Stütze ist in diesem Beispiel in einer Instanz der Klasse *IfcRectangleProfileDef* enthalten. Um diese zu variieren, wird ein Zeiger (*IfcAttributeBinding*) instanziiert, der auf das betreffende Attribut des *IfcRectangleProfileDef*-Objekts zeigt. Dazu wird eine Relation zwischen den beiden Instanzen über das *BoundObject*-Attribut gesetzt. Zusätzlich wird als String der Name des Attributes im *BoundAttribute* hinterlegt. Der Wert des *BoundIndex* ist unbesetzt, da *XDim* kein Aggregationsdatentyp ist. Die Variationswerte befinden sich im Listenattribut *VarianceList* einer *IfcAttributeVariance*-Instanz. Über die *IfcVarianceConnector* werden die Variationswerte mit dem Zeiger verknüpft. Auf die eben beschriebene Weise können auch weitere Variationen im Gebäudemodell abgelegt werden.

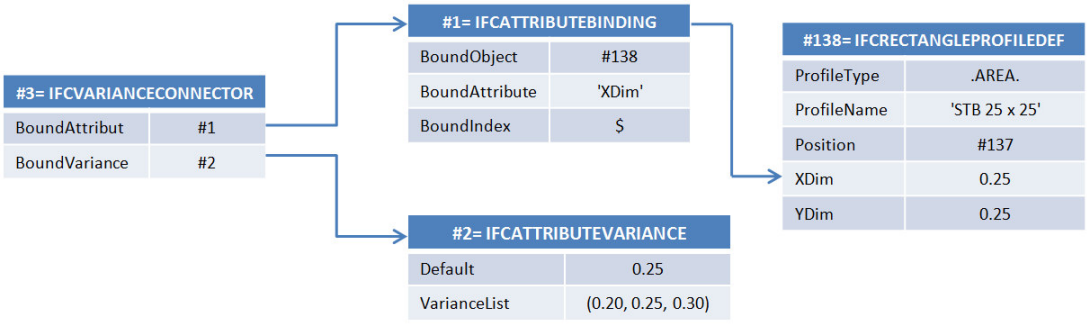


Abbildung 4.8: Parametervariation der Stützenbreite im STEP (schematisch)

Für Variationen, die beispielsweise den kompletten Querschnitt eines Trägers betreffen, müssen mehrere Parametervariationen miteinander kombiniert werden. Bei einem Rechteck-Profil betrifft das die Breite und Höhe der Grundfläche bzw. das *XDim* und *YDim*-Attribut einer *IfcRectangleProfileDef* im IFC-Gebäudemodell. Um Variation dieser Art ebenfalls zu ermöglichen, muss der Ansatz des Variationsmodells erweitert werden. Dazu werden die Parametervariationen in übergestellten Variationsgruppen zusammengefasst. Der neu eingeführte Datentyp *IfcModellVariation* soll diese Gruppen repräsentieren und besitzt ein Set (sortierte Liste ohne Dopplungen) aus mindestens einer Variation im Attribut *Variation* (s. Listing 4.5). Mit dieser Struktur sind bereits Kombinationen möglich, jedoch soll zusätzlich zwischen verschiedenen Kombinationsarten unterschieden werden können. In Anlehnung an die MMQL nach [Fuc15] und die SQL nach [ISOe] sollen drei verschiedene Kombinationsarten unterschieden und an einem Beispiel mit den zwei Variationsmengen $A = \{a, b, c, d\}$ und $B = \{x, y, z\}$ erläutert werden:

- **Inner Join**: Die Werte einer sortierten Liste werden mit den Werten der anderen Listen in Abhängigkeit ihrer Position zu einer Menge aus n-Tupeln verknüpft (n ist die Anzahl der zu kombinierenden Variationen). Dabei werden in der Ergebnismenge unvollständige Tupel weggelassen. Die Ergebnismenge aus der Operation zwischen den Variationsmengen A und B wird im Folgenden aufgeführt:

  $$A \text{ INNER JOIN } B = \{(a, x), (b, y), (c, z)\}$$

  Die Inner Join-Operation richtet sich nach der Variation mit der geringsten Anzahl an Werten. Der Wert d aus der Menge A entfällt, da kein Gegenwert an der vierten Stelle der Menge B vorhanden ist.

- **Outer Join**: Die Werte einer sortierten Liste werden mit den Werten der anderen Listen in Abhängigkeit ihrer Position zu eine Menge aus n-Tupeln verknüpft (n ist die Anzahl der zu kombinierenden Variationen). Dabei werden in der Ergebnismenge unvollständige Tupel dennoch aufgeführt. Die fehlenden Werte werden mit dem Default-Wert der jeweiligen Menge ergänzt[6]. Die Ergebnismenge aus der Operation zwischen den Variationsmengen A und B zeigt die folgende Gleichung:

  $$A \text{ OUTER JOIN } B = \{(a, x), (b, y), (c, z), (d, x)\}$$

  Die OUTER Join-Operation richtet sich nach der Menge mit der größten Anzahl an Werten. Der Wert d aus der Menge A wird mit dem Defaultwert x der Menge B gekoppelt, da letzteres kein viertes Element besitzt.

[6] in diesem Beispiel sind die Default-Werte die ersten Werte der jeweiligen Menge. Im Variationsmodell sind sie im *Default*-Attribut der Klasse *IfcAttributVariance* hinterlegt.

- **Cross Join**: Auch als "Kartesisches Produkt" bekannt, werden aus n verschiedenen Mengen eine Ergebnismenge aus allen geordneten n-Tupeln von Elementen der Mengen gebildet. Die Ergebnismenge aus der Operation zwischen den Variationsmengen A und B ist nachfolgend aufgeführt:

$$A \text{ CROSS JOIN } B = \{(a,x),(a,y),(a,z),(b,x),(b,y),(b,z),(c,x),\ldots \\ \ldots(c,y),(c,z),(d,x),(d,y),(d,z)\}$$

Die aufgeführten Kombinationsarten bzw. Verbundoperationen werden im Variationsmodell gespeichert. Das Ergebnis wird erst für die Bildung der Modellinstanzen beim Interpreter ermittelt. Dazu wird die Enumeration *IfcJoinOperator* eingeführt (s. Listing 4.5). Deren Enumerationswerte sind *CROSS*, *INNERJOIN* und *OUTERJOIN*, welche die drei zuvor aufgeführten Verbundoperationen Cross Join, Inner Join und Outer Join repräsentieren. Zudem wird das Attribut *JoinMode* vom Datentyp *IfcJoinOperator* in der Klasse *IfcModellVariation* ergänzt. So kann für die einzelnen Variationsgruppen ein jeweiliger Verbundoperator gewählt werden.

Listing 4.5: IfcModellvariance und IfcJoinOperator in EXPRESS

```
1  ENTITY  IfcModellVariation;
2  JoinMode:        IfcJoinOperator;
3  Variation:       SET [1:?] OF IfcVarianceConnector;
4  END_ENTITY;
5
6  TYPE IfcJoinOperator = ENUMERATION OF
7  (Cross,
8  InnerJoin,
9  OuterJoin);
10 END_TYPE;
```

Die Zusammenfassung zu übergeordneten Variationsgruppen und die Bereitstellung verschiedener Verbundoperationen ermöglichen die gleichzeitige Ausführung von Variationen. Damit geht einher, dass der Interpreter nur die Variationsgruppen behandelt. Diese Vorgehensweise ist notwendig, da die einzelnen Variationen möglicherweise bereits in den übergestellten Gruppen verarbeitet sind. Einzelne Parametervariationen müssen deshalb ebenfalls in eine Instanz von *IfcModellVariation* verpackt werden. So muss die STEP-Umsetzung für das vorangegangene Beispiel (s. Abb. 4.8) entsprechend geändert werden. Die Anpassungen an dem Beispiel sind in Abbildung 4.9 zu sehen.

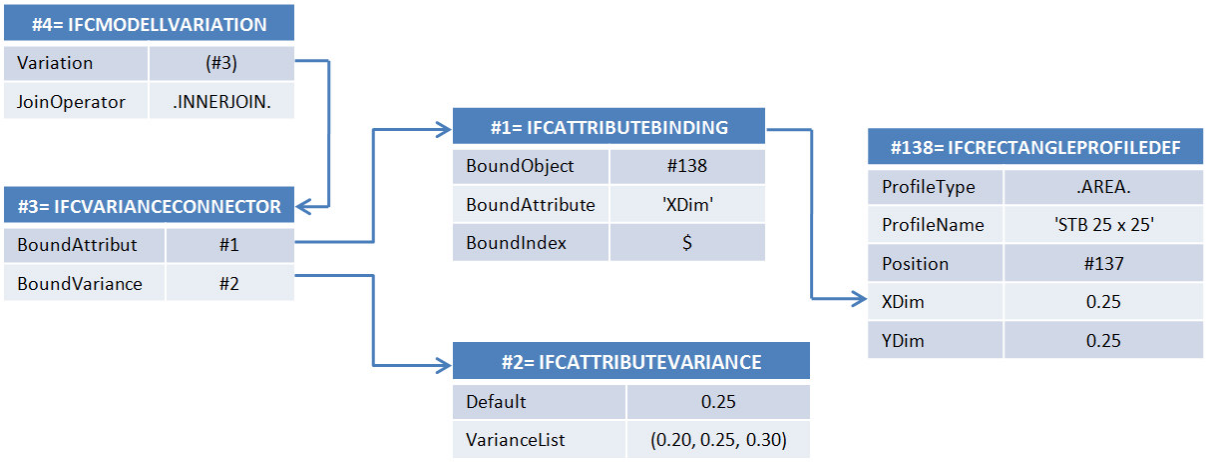


Abbildung 4.9: Modellvariation der Stützenbreite in STEP (schematisch)

Als weiteres Exempel soll die bereits erwähnte Variation von Stützenbreite und -höhe aufgeführt werden. Dazu müssen jeweils zwei Zeiger (*IfcAttributeBinding*) auf die Attribute *XDim* und *YDim* des Rechteckprofils (*IfcRectangleProfileDef*) instanziiert werden. Hinzu kommen die Variationen der jeweiligen Parameter. In diesem Beispiel wurden zwei Instanzen für die Variationswerte genutzt. Da diese jedoch identisch sind, wäre die Nutzung einer Variationsliste für beide Parameter möglich gewesen. Mit den Instanzen von *IfcVarianceConnector* werden die Variationslisten mit dem jeweiligen Zeiger verknüpft und durch *IfcModellVariation* zu einer mehrparametrigen Variation zusammengefasst. Abbildung 4.10 zeigt die soeben beschriebenen Spezifikationen. Das Ergebnis der komplexen Parametervariation zeigt Tabelle 5.1 mit den Parameterbelegungen für die gebildeten Modellinstanzen.

Tabelle 4.6: Variationen der Stützenbreite und -Höhe nach Abb. 4.10

Tochtermodellnr.	Stützenbreite	Stützenhöhe
1	0.20	0.20
2	0.20	0.25
3	0.20	0.30
4	0.25	0.20
5	0.25	0.25
6	0.25	0.30
7	0.30	0.20
8	0.30	0.25
9	0.30	0.30

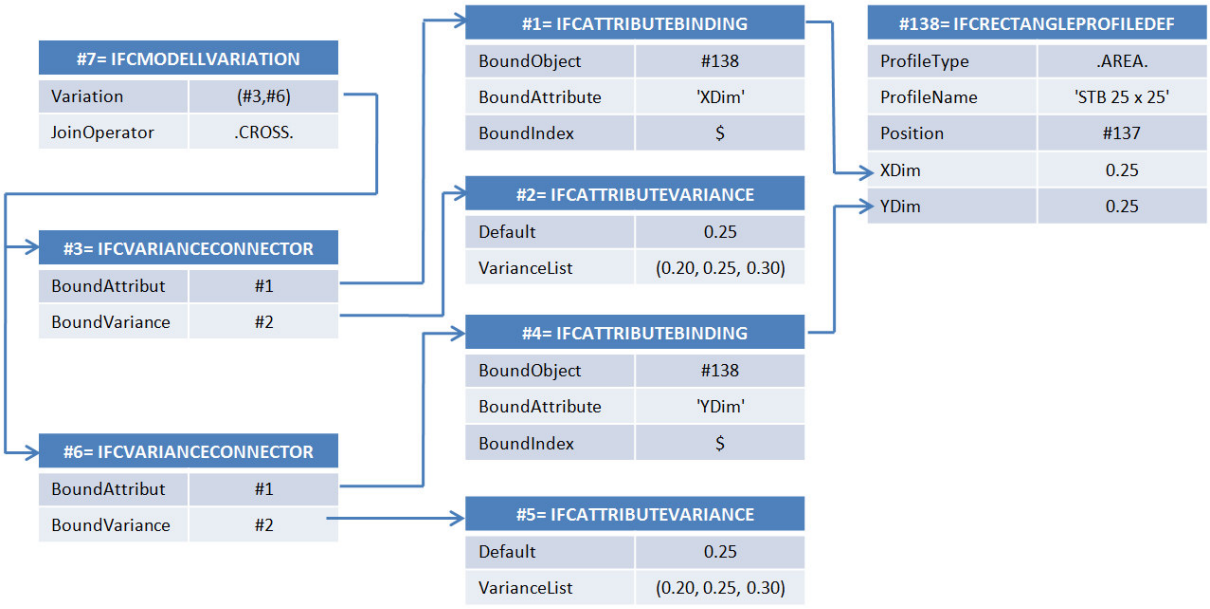


Abbildung 4.10: Variation der Stützenbreite und -Höhe in STEP (schematisch)

Abbildung 4.11 zeigt den vollständigen Ansatz des Variationsmodells in EXPRESS-G. Im Anhang A.1 kann das Schema im reinen EXPRESS eingesehen werden. Das Variationsmodell erfüllt alle in Kapitel 4.1.1 geforderten Randbedingungen. So ist die Implementierung und Nutzung des Variationsmodells rein optional und kann auch für bereits bestehende Gebäudemodelle hinzugezogen werden. Die Variationen werden im Modell an die entsprechenden Attribute gebunden. Sollten komplexere Variationen notwendig werden, ermöglicht dieser Ansatz zahlreiche Möglichkeiten zu deren Erstellung. Es können intern Variationswertelisten durch mehrere Mengenoperationen manipuliert und angepasst werden. Dazu gehört beispielsweise der Ausschluss von Werten oder deren Vereinigung. Extern können die einzelnen Parametervariationen in übergeordnete Variationsgruppen mithilfe von Verbundoperationen zusammengefügt werden. Diese Lösung erlaubt beispielsweise Variationen von kompletten Bauteilquerschnitten oder Koordinatenpunkten.

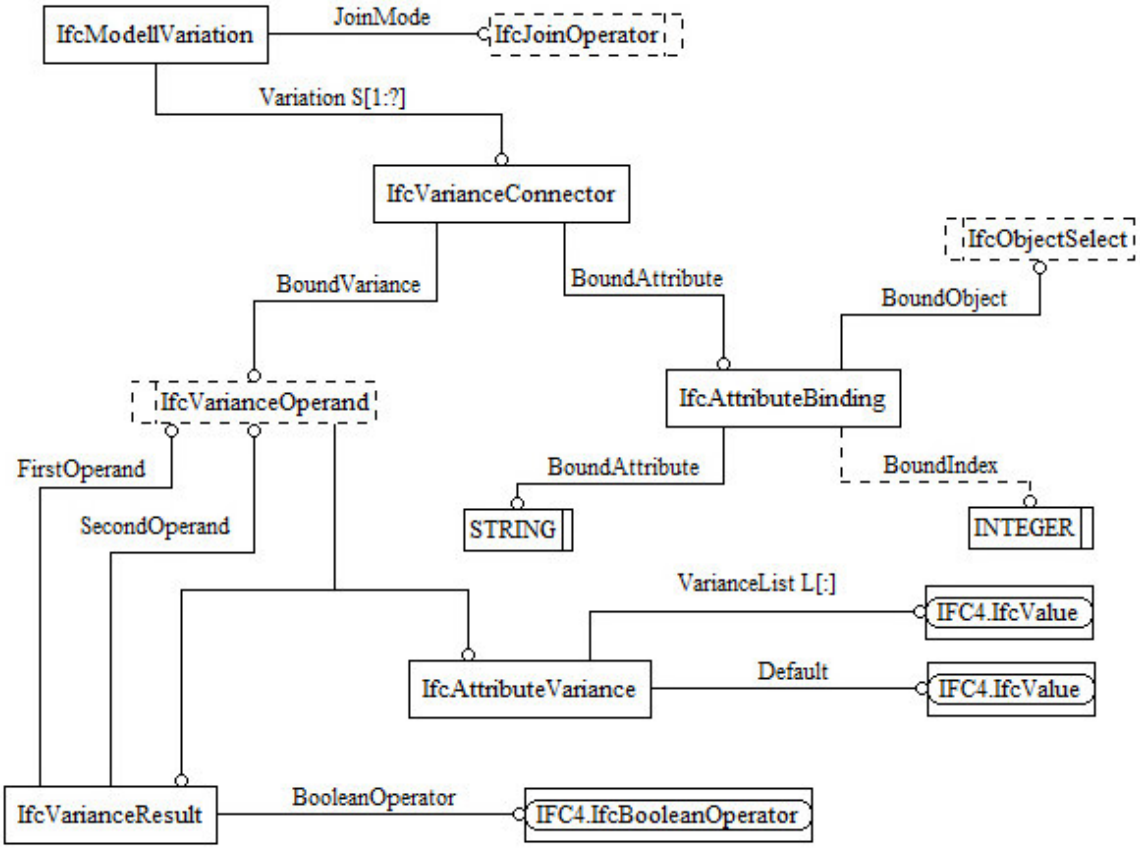


Abbildung 4.11: Komplettes Schema des Variationsmodells in EXPRESS-G

5 Verifikation des Konzepts zur Definition von Parametervariationen

Im vorangegangenen Kapitel 4 wurde ein möglicher Ansatz für die Implementierung von Parametervariationen in IFC erarbeitet und diskutiert. Mit dem erstellten Variationsschema können multiple Parametervariationen, die in Variationsgruppen zusammengefasst sind, in das Gebäudemodell integriert werden. Die Generierung der Modellinstanzen soll mit einem Interpreter erfolgen. In diesem Abschnitt wird für das Variationsschema ein solcher Interpreter vorgestellt, welcher im Rahmen dieser Arbeit entwickelt wurde. Gleichzeitig soll mit dessen Hilfe das Konzept der Parametervariation an verschiedenen Beispielen getestet und verifiziert werden.

5.1 Entwicklung einer Java-Anwendung zur Variation eines IFC-Gebäudemodells

Der Interpreter für das Variationsmodell wurde mithilfe der Entwicklungsumgebung *Eclipse* in Java erstellt. Mit Hilfe dieser Anwendung sollen die Modellinstanzen mit den variierten Parametern aus den Variationsmodellen und den zugehörigen IFC-Gebäudemodellen generiert werden. Das Variationsmodell wird als eine externe Erweiterung des IFC-Schemas angesehen, wodurch sich die Daten für die Parametervariation und die des Gebäudemodells in zwei unterschiedlichen STEP-Dateien befinden.

5.1.1 Grafische Benutzeroberfläche des Interpreters

Um dem Anwender die Nutzung des Interpreters zu erleichtern, wurde eine grafische Benutzeroberfläche erstellt. Das Graphical User Interface (GUI) wird in Abbildung 5.1 gezeigt und gliedert sich in Eingabemaske, Startknopf für die Prozedur und Ausgabefenster.

In der Eingabemaske werden die notwendigen Daten für die Prozedur mit der Angabe des dazugehörigen Dateipfades eingegeben. Der Nutzer wählt dazu das Gebäudemodell (*.ifc) und das Variationsmodell (*.p21) im Dateibrowser (s. Abbildung 5.2), der sich beim Betätigen des ”Durchsuchen”-Buttons öffnet, aus. Alternativ kann der Dateipfad zu den Modellen auch direkt in den vorgegebenen Textfeldern eingegeben werden. Dateien mit invaliden Dateiendungen für die entsprechende Auswahl werden dabei ausgeblendet. Mit dem Choice-Dialogelement kann außerdem die verwendete IFC-Version ausgewählt werden.

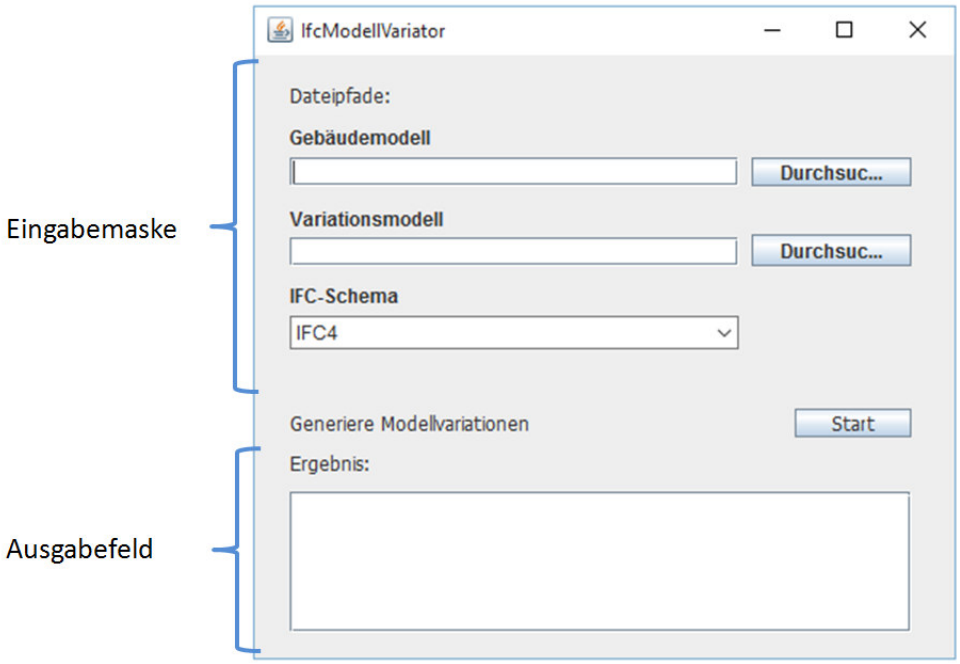


Abbildung 5.1: GUI des Interpreters für Parametervariationen

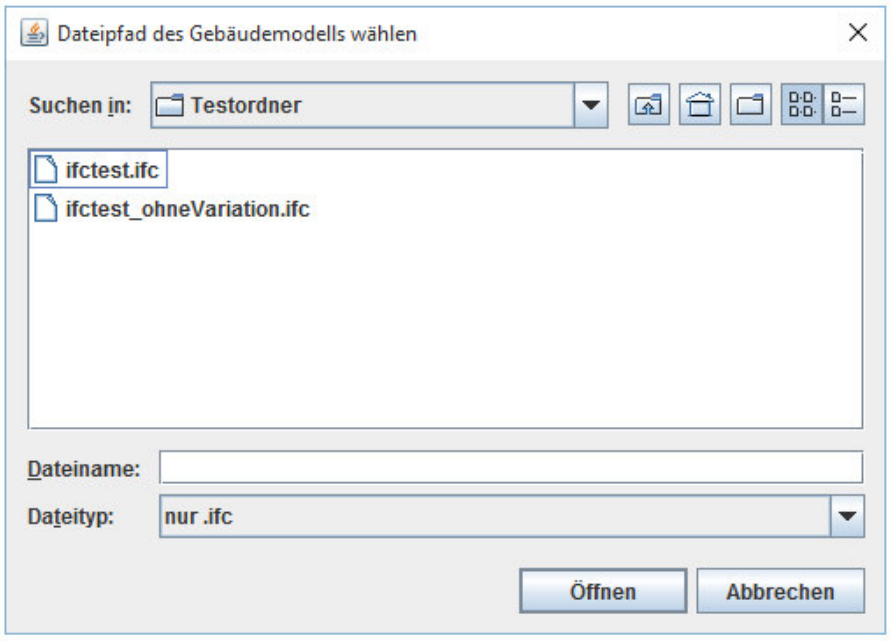


Abbildung 5.2: Fenster zur Auswahl des Dateipfades

Sind die notwendigen Daten vorhanden kann die Prozedur für die Modellvariation mit dem ”Start”-Button gestartet werden. Im darunter platzierten Textfeld werden Meldungen zum aktuellen Stand des Prozesses ausgegeben. Auch werden Fehlermeldungen über das Textfeld übermittelt. Die Textausgaben dokumentieren den Programmablauf. Treten während der Prozedur keine Fehler auf, so befinden sich die generierten Modellinstanzen im angegebenen Zielordner.

5.1.2 Struktur des Interpreters

Der Interpreter für die Variation von Gebäudemodellen auf Basis von IFC-Daten wurde mit einem imperativen Programmieransatz umgesetzt. Ein objektorientierter Ansatz wäre durch die Verwendung des IFC-Schemas ebenfalls möglich gewesen. Jedoch müssen dabei alle IFC-Klassen in Java-Klassen umgesetzt werden. Damit müssen auch alle Instanzen in der STEP-Datei im Interpreter instanziiert werden. Das hat einen hohen Speicherverbrauch und eine umständliche Handhabung der vielen Objektinstanzen zur Folge. Den Nutzen, den ein objektorientierter Ansatz für den Interpreter mit sich bringen würde, wiegt dessen Nachteile damit nicht auf. Der Interpreter setzt sich aus den folgenden fünf Klassen zusammen:

- VarianceModellFrame
- Program
- VariationSet
- CustomOutputStream
- FileExtensionFilter

Die Klasse *VarianceModelFrame* repräsentiert die grafische Benutzeroberfläche des Interpreters (s. Abschnitt 5.1.1), über die der Nutzer die notwendigen Eingaben für die Parametervariationen eines Gebäudemodells machen kann. Die GUI ist Event-gesteuert. Mit ihr wird die Prozedur für die Generierung der Modellinstanzen ausgelöst. Sie bildet damit die Schnittstelle zur Nutzung des Interpreters und den Startpunkt des Programms. Dazu besitzt die Klasse die Methode *initialize()*, welche die GUI initialisiert. Abbildung 5.3 zeigt das Klassendiagramm von *VarianceModelFrame* in der Unified Modeling Language (UML).

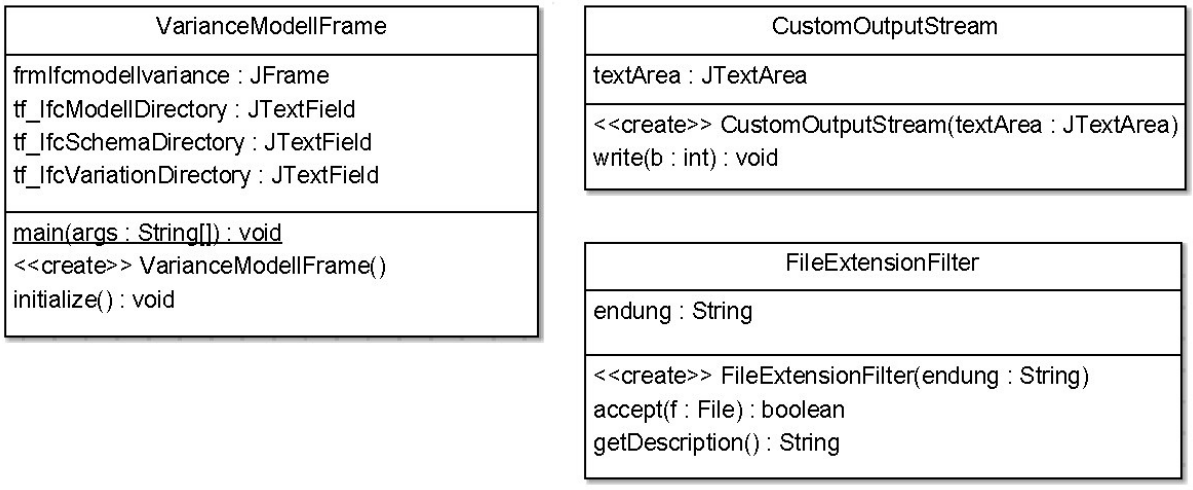


Abbildung 5.3: *VarianceModellFrame*, *FileExtensionFilter* und *CustomOutputStream* im UML-Klassendiagramm

Die Klassen *FileExtensionFilter* und *CustomOutputStream* (s. Abb. 5.3) sind Hilfsklassen für das *VarianceModellFrame*. Der *FileExtensionFilter* ist ein Datenfilter und verhindert die Eingabe

von Dateien im falschem Format bzw. mit falschen Dateiendungen auf der GUI. Fehler bei der Dateneingabe können somit größtenteils verhindert werden. Die Klasse *CustomOutputStream* ist ein Ausgabedatenstrom, welcher Strings an ein Textfeld der *VarianceModellFrame* ausgibt. Die Ausgabe gibt Auskunft zum aktuellen Stand der Prozedur sowie Fehlermeldungen beim Auftreten eines Fehlers. Die Klassen *FileExtensionFilter* und *CustomOutputStream* wurden aus [V.a] und [V.b] entnommen.

Der Hauptteil des Interpreters bildet die Klasse *Program* (s. Abb. 5.4). Sie enthält die prozeduralen Anweisungen zur Bildung der Modellinstanzen aus dem Variationsmodell. Für die Ausführung der Anweisung werden bei der Initialisierung der Klasseninstanz die Dateipfade, für das Gebäudemodell in IFC, das dazu verwendete IFC-Schema, das dazugehörige Variationsmodell, sowie der Dateipfad des Zielordner für die Modellinstanzen, von der *VarianceModelFrame*-Instanz übergeben. Die *execute()*-Methode startet die Prozedur mit dem Einlesen des Gebäudemodells und des verwendeten IFC-Schemas. Mit der Methode *ifcDataReader()* wird das IFC-Gebäudemodell eingelesen. Danach wird das Variationsmodell, welches als STEP-Datei vorliegt, nach allen Instanzen der Klasse *IfcModellVariance* durchsucht und deren STEP-ID der ArrayList *ifcVarianceList* hinzugefügt. Die STEP-Dateien werden zeilenweise eingelesen und in einer ArrayList zwischengespeichert. Sind die STEP-Dateien komplett eingelesen, werden die ArrayLists, in der die Dateien zwischengespeichert sind, für eine schnellere Verarbeitung in ein Array umgewandelt und den Variablen *ifcData* (Gebäudemodell) und *ifcVariation* (Variationsmodell) übergeben. Auf ähnliche Weise wird das IFC-Schema mit der Methode *ifcSchemaReader()* eingelesen und im String Array *ifcSchema* abgelegt. Danach werden die in *ifcVarianceList* abgelegten Modellvariationen mit der Methode *modelVariance()* abgearbeitet und deren Modellinstanzen gebildet und im Zielordner abgespeichert. Dies geschieht in zwei groben Teilschritten. Zunächst werden alle notwendigen Variationsdaten aller betroffenen Parametervariationen aus dem IFC-Schema und des Gebäudemodells extrahiert und in die zweidimensionalen Arrays *binding* und *variance* und dem String *mode* eingefügt. Das Array *variance* enthält die Variationswerte der Parametervariation, welche durch die rekursive Methode *getVariance()* ermittelt werden. An dieser Stelle werden auch, falls vorhanden, die Mengenoperationen der Variationswerte mit der Klasse *VariationSet* und der Methode *getBooleanResult()* verarbeitet. Die Klasse *VariationSet* wird im späteren noch genauer erläutert. Im Array *binding* werden die Daten des Variationsparameters abgelegt. Sie enthält die STEP-ID, die Position des Attributes, und optional die Indexnummer des Parameters. Die Variable *mode* enthält den Verbundoperator (InnerJoin, OuterJoin, Cross; s. dazu auch Abschnitt 4.3.3) der Modellvariation und wird im zweiten Teilschritt, dem Erstellen der Modellinstanzen, zur Fallunterscheidung genutzt. Für die Modellerstellung wird zunächst die Anzahl der zu bildenden Modellinstanzen ermittelt. Daraufhin wird jede Instanz einzeln über eine Schleife zeilenweise geschrieben. Hauptsächlich wird dabei das ursprüngliche Gebäudemodell in STEP kopiert. Ist jedoch eine Objektdeklaration ein Variationsparameter, so wird die entsprechende Deklaration über die Methode *getNewLine()* modifiziert. So werden alle Modellinstanzen gebildet und im Zielordner abgespeichert. Die Anzahl der zu bildenden Modellinstanzen sowie die Art der Verknüpfung der Variationswerte von mehreren Parametervariationen, innerhalb einer Modellvariation, werden durch die Fallunterscheidung der Verbundoperation gesteuert.

Abbildung 5.4: *Program* im UML-Klassendiagramm

Der komplette Ablauf für die Erstellung der Modellinstanzen sind im Anhang B als UML-Aktivitätsdiagramme visualisiert.

Die Klasse *VariationSet* repräsentiert die Variationsmenge einer Parametervariation. Sie ermöglicht mit ihrem Konstruktor die Transformation der Wertelisten in Wertemengen einer Parametervariation. Die Methoden *union()*, *intersection()* und *difference()* entsprechen den Mengenoperationen Vereinigung, Durchschnitt und Differenz. Diesen Methoden werden bei Aufruf aus einer *VariationSet*-Instanz heraus die zweite Variationsmenge übergeben. Ihr Rückgabewert entspricht dem Ergebnis der entsprechenden Mengenoperation aus der aufrufenden und übergebenen Variationsmenge. Daneben verfügt die Klasse noch über die Methode *toArray()*, welche eine Rücktransformation der Menge zu einer Liste ermöglicht. Anzumerken ist dabei, dass zuvor entfernte Dopplungen der Wertelemente nicht wiederhergestellt werden, sondern lediglich eine Reihenfolge für die Elemente eingeführt wird (sie werden in aufsteigender Reihenfolge sortiert). Die Methode dient der Weiterverarbeitung der Daten. Das Klassendiagramm der Klasse *VariationSet* zeigt die Abbildung 5.5.

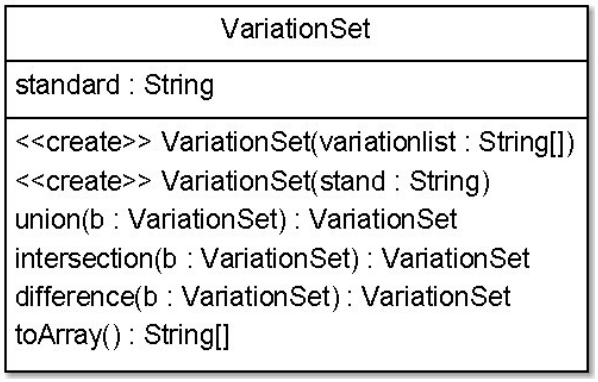


Abbildung 5.5: *VariationSet* im UML-Klassendiagramm

5.2 Verifikation des Konzepts

Für die Verifikation des entwickelten Ansatzes zur Einführung von Parametervariationen in die IFC wurde das Konzept an einem Beispiel getestet. Parallel dazu wurde ebenfalls der Interpreter für Parametervariation, der im Rahmen dieser Arbeit entwickelt wurde (s. Abschnitt 5.1), validiert. Im Anschluss soll die Generalisierbarbeit des Konzepts gezeigt werden.

5.2.1 Beispiel: Betondecke

In diesem Beispiel soll die Höhe einer Betondecke variiert werden, um die Funktionsweise des Variationsmodells zu zeigen. Die Deckenhöhe beträgt im IFC-Gebäudemodell 20 cm. Es sollen drei Modellinstanzen mit den Deckenhöhen 10, 30 und 40 cm erstellt werden. Für die Erstellung existieren verschiedene Möglichkeiten. Jede Modellinstanz kann in einer separaten Modellvariation definiert werden. Es ist jedoch ressourcenschonender sie in nur einer Modellvariation zu definieren. Dazu wird eine Parametervariation für die Deckenhöhe erstellt und die Werte 10, 30 und 40 als ihre Parameterwerte angegeben. Alternativ können zur Angabe der Werte auch Mengenoperationen verwendet werden. Diese Variante erhält jedoch erst bei einer automatisierten Erstellung von Parametervariationen eine größere Bedeutung. Das folgende Listing wurde für die Generierung der geforderten Modellinstanzen verwendet:

Listing 5.1: Variationsmodell für Bsp. Betondecke

```
1 DATA;
2 #200= IFCMODELLVARIATION((#210),.INNERJOIN.);
3 #210= IFCVARIANCECONNECTOR(#220,#230);
4 #220= IFCATTRIBUTEVARIANCE(0.2,(0.1,0.3,0.4));
5 #230= IFCATTRIBUTEBINDING(#143,'DEPTH',$);
6 ENDSEC;
```

Darin wird die Parametervariation für die Deckenhöhe in der Instanz #210 definiert. Ihr Attribut *BoundVariance* ist mit der Instanz #220 belegt, welche vom Typ *IfcAttributeVariance* ist und einen Defaultwert von 0.2 und die Parameterwerte 0.1, 0.3 und 0.4 (in Metern) besitzt. Mit der

Instanz #230 wird das Attribut *BoundAttribute* der Parametervariation belegt und beschreibt den Zeiger zur Höhe der Decke im Ausgangsmodell. #143 ist der Verweis zum zu variierenden Objekt im Gebäudemodell. Dieser ist vom Typ *IfcExtrudedAreaSolid*, dessen Attribut *DEPTH* die Betondeckenhöhe definiert. Die Parametervariation (#210) wird für die Auswertung in eine Modellvariation (#200) verpackt. Abbildung 5.6 zeigt das Ausgangsmodell und die, aus der Variation resultierenden, Modellinstanzen mit der geforderten Parameterbelegung. Die zweite Variante eines Variationsmodells unter Verwendung einer Mengenoperation ist im Listing 5.2 aufgeführt und generiert die gleichen Modellinstanzen.

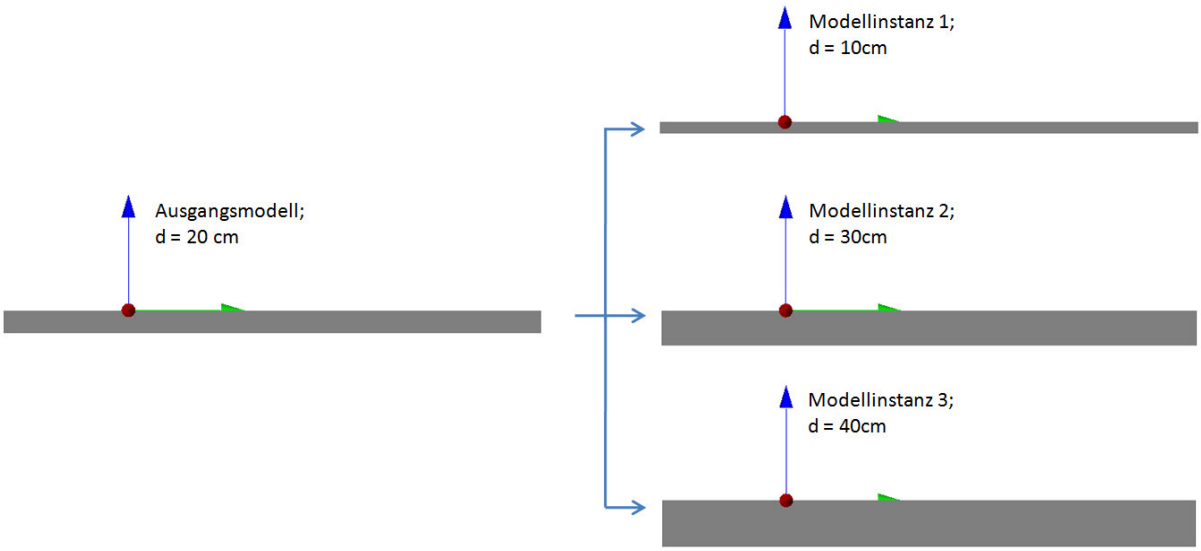


Abbildung 5.6: Ausgangsmodell und generierte Modellinstanzen des Bsp. Betondecke

Listing 5.2: Variationsmodell für Bsp. Betondecke (Variante 2)

```
1 DATA;
2 #200= IFCMODELLVARIATION((#210),.INNERJOIN.);
3 #210= IFCVARIANCECONNECTOR(#220,#230);
4 #220= IFCVARIANCERESULT(#221,#222,.DIFFERENCE.);
5 #221= IFCATTRIBUTEVARIANCE(0.2,(0.1,0.2,0.3,0.4));
6 #222= IFCATTRIBUTEVARIANCE(0.2,(0.2));
7 #230= IFCATTRIBUTEBINDING(#143,'DEPTH',$);
8 ENDSEC;
```

5.2.2 Beispiel: 3 Stützen

Das Beispiel "3 Stützen" (vgl. Abb. 5.7) ist ein Minimalbeispiel, bestehend aus drei aufgereihten Rundstützen mit wachsenden Durchmessern. Die Stützen befinden sich auf einer Ebene und besitzen die gleiche Höhe von zwei Metern. Ihre Durchmesser betragen in aufsteigender Reihenfolge 20, 30 und 40 cm.

An diesem Beispiel werden die verschiedenen Verbund- und Mengenoperationen des Variationsmodells aufgezeigt und geprüft. Zur besseren Veranschaulichung werden hierbei nur die Höhen der drei Stützen variiert. Für die Variation der einzelnen Säulenhöhen wird dabei jeweils eine Mengenoperation für die Angabe der Variationswerte verwendet. Die Parametervariationen werden zudem mit den unterschiedlichen Verbundoperationen zu Modellvariationen verknüpft. Listing 5.3 zeigt das dazu verwendete Variationsmodell in STEP.

Listing 5.3: Variationsmodell für Bsp. 3 Stützen in STEP

```
DATA;
#500= IFCMODELLVARIATION((#510,#540,#570),.INNERJOIN.);
#501= IFCMODELLVARIATION((#510,#540,#570),.OUTERJOIN.);
#502= IFCMODELLVARIATION((#510,#540,#570),.CROSS.);
#510= IFCVARIANCECONNECTOR(#520,#530);
#520= IFCVARIANCERESULT(#521,#522,.UNION.);
#521= IFCATTRIBUTEVARIANCE(2.0,(1.0,1.5));
#522= IFCATTRIBUTEVARIANCE(2.0,(2.0,2.5));
#530= IFCATTRIBUTEBINDING(#142,'DEPTH',$);
#540= IFCVARIANCECONNECTOR(#550,#560);
#550= IFCVARIANCERESULT(#551,#552,.INTERSECTION.);
#551= IFCATTRIBUTEVARIANCE(2.5,(1.5,2.0,2.5,3.0,3.5));
#552= IFCATTRIBUTEVARIANCE(2.0,(1.5,2.5,3.5));
#560= IFCATTRIBUTEBINDING(#328,'DEPTH',$);
#570= IFCVARIANCECONNECTOR(#580,#590);
#580= IFCVARIANCERESULT(#581,#582,.DIFFERENCE.);
#581= IFCATTRIBUTEVARIANCE(3.0,(2.0,2.5,3.0));
#582= IFCATTRIBUTEVARIANCE(2.0,(1.5,2.0,2.5));
#590= IFCATTRIBUTEBINDING(#433,'DEPTH',$);
ENDSEC;
```

Die Parametervariation der ersten Stütze wird mit den Instanzen #510 bis #530 beschrieben. Ihre Parameterwerte (#520) entstehen aus der Vereinigung aus den zwei Wertelisten (#521,#522). Die Variationen der anderen beiden Stützen werden mit den Instanzen #540 bis #560 und #570 bis #590 definiert. Ihre Parameterwerte werden auf ähnliche Weise definiert, jedoch werden für diese andere Mengenoperationen verwendet. Die Werte der zweiten und dritten Stütze werden aus den Operationen Durchschnitt und Differenz gebildet (s. dazu auch Abschnitt 4.3.2). Tabelle 5.1 zeigt die resultierenden Parameterwerte für die Variationen der einzelnen Stützen und veranschaulicht deren Erstellung. Im Variationsmodell werden, aus den soeben beschriebenen Parametervariationen, drei Modellvariationen definiert (#500,#501 und #502), bei denen jeweils einer der drei Verbundoperatoren INNER JOIN, OUTER JOIN und CROSS zum Einsatz kommt. Die daraus entstehenden Modellinstanzen werden mit ihren spezifischen Parameterbelegungen in der Tabelle 5.2 aufgeführt. Zur besseren Veranschaulichung befinden sich im Anhang C zusätzlich die Abbildungen aller generierten Modellinstanzen.

Tabelle 5.1: Variationswerte des Bsp. 3 Stützen

	Höhe der		
	1. Säule	2. Säule	3. Säule
1. Variationsliste	(1.0,1.5)	(1.5,2.0,2.5,3.0,3.5)	(2.0,2.5,3.0)
Default-Wert	2.0	2.5	3.0
2. Variationsliste	(2.0,2.5)	(1.5,2.5,3.5)	(1.5,2.0,2.5)
Default-Wert	2.0	2.0	2.0
Mengenoperation	UNION	INTERSECTION	DIFFERENCE
Ergebnisliste	(1.0,1.5,2.0,2.5)	(1.5,2.5,3.5)	(3.0)
Default-Wert	2.0	2.5	3.0

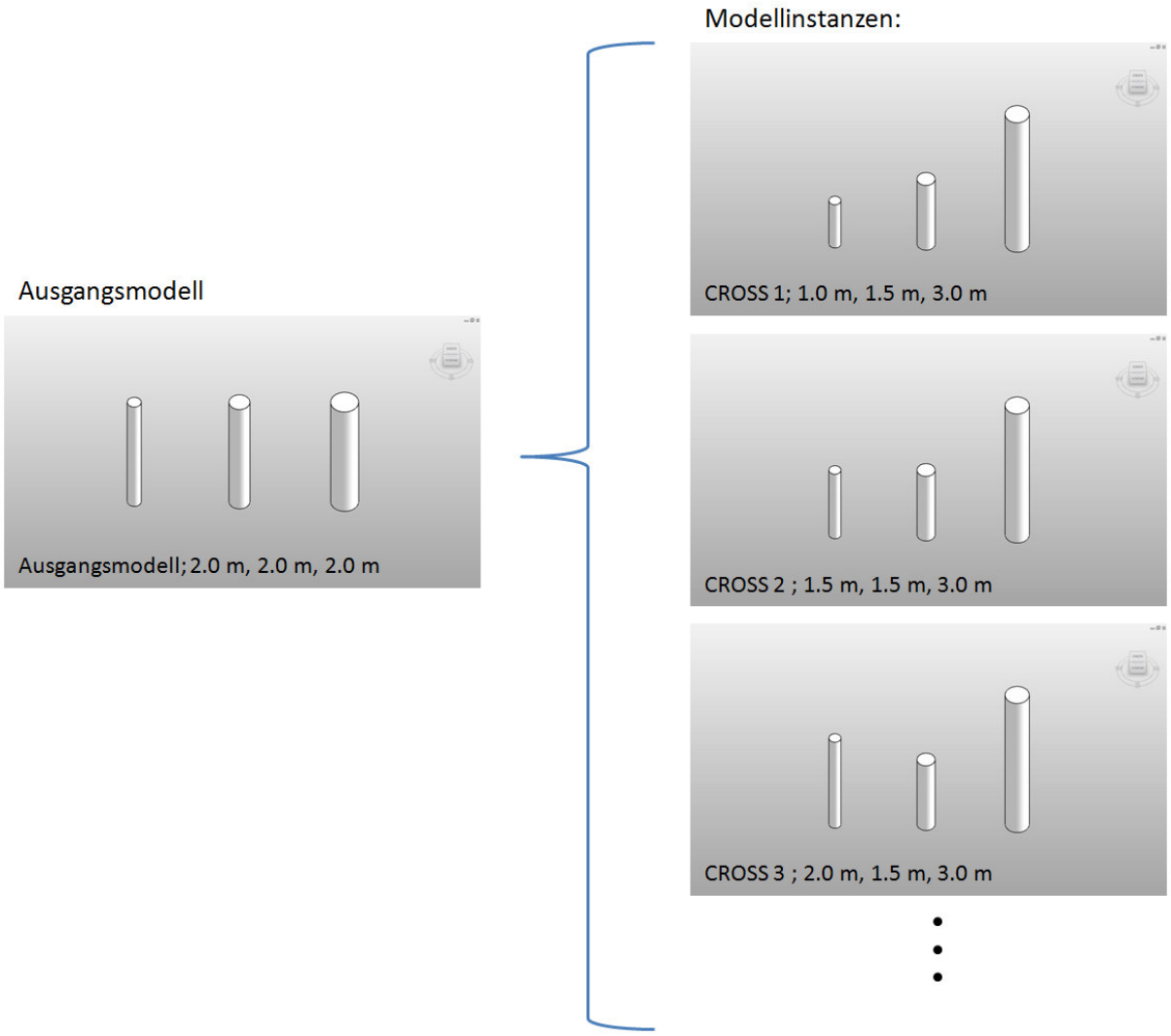


Abbildung 5.7: 3D-Visualisierung von Bsp. 3 Stützen

Tabelle 5.2: Modellinstanzen des Bsp. 3 Stützen

Verbund-Operation	Tochter-Modellnr.	Höhe der 1. Säule	2. Säule	3. Säule
INNER JOIN	1	1.0	1.5	3.0
OUTER JOIN	1	1.0	1.5	3.0
	2	1.5	2.5	3.0
	3	2.0	3.5	3.0
	4	2.5	2.5	3.0
CROSS	1	1.0	1.5	3.0
	2	1.5	1.5	3.0
	3	2.0	1.5	3.0
	4	2.5	1.5	3.0
	5	1.0	2.5	3.0
	6	1.5	2.5	3.0
	7	2.0	2.5	3.0
	8	2.5	2.5	3.0
	9	1.0	3.5	3.0
	10	1.5	3.5	3.0
	11	2.0	3.5	3.0
	12	2.5	3.5	3.0

5.2.3 Generalisierbarkeit des Konzepts

Für die Verifikation des Konzepts der Parametervariation auf Basis von IFC-Daten wurden die Parametervariationen mithilfe des STEP-Formats und des Variationsschemas in ein Variationsmodell integriert. Das Konzept ist jedoch nicht nur auf diese beschränkt, sondern kann auch auf andere Auszeichnungs- und Modellierungssprachen, wie z. B. SGML,XML und XSD, angewendet werden. Wie im Abschnitt 4.1.1.4 bereits beschrieben, bietet die Nutzung von EXPRESS und STEP in Verbindung mit der IFC besondere Vorteile, weswegen diese als Vorzugsvariante gewählt wurde. Um die Generalisierbarkeit des Konzepts zu zeigen, wurde das EXPRESS-Schema des Variationsmodells in ein XML-Schema umgeformt. Ein Ausschnitt des XML-Schema zeigt folgendes Listing:

Listing 5.4: Transformation von IfcModellvariation aus EXPRESS zu XML Schema

```
// EXPRESS-Definition
ENTITY  IfcModellVariation;
JoinMode        :       IfcJoinOperator;
Variation       :       SET [1:?] OF IfcVarianceConnector;
END_ENTITY;

// XML Schema-Definition
<xs:element name="IfcModellVariation" type="
   IfcModellVariation"  nillable="true"/>
<xs:complexType name="IfcModellVariation">
 <xs:complexContent>
  <xs:sequence>
   <xs:element name="Variation">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="IfcVarianceConnector" minOccurs="1"
         maxOccurs="unbounded"/>
     </xs:sequence>
    <xs:attribute ref="ifc:itemType" fixed="
       IfcVarianceConnector"/>
    <xs:attribute ref="ifc:cType" fixed="set"/>
    <xs:attribute ref="ifc:arraySize" use="optional"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="JoinMode" type="IfcJoinOperator" use="
     optional"/>
 </xs:complexContent>
</xs:complexType>
```

Die Transformation des Variationsmodells erfolgte gemäß der allgemeinen Methode zur Umwandlung von EXPRESS- zu XML- Schemata die unter [ISOc] beschrieben ist. Das komplette Variationsmodell in XSD kann im Anhang A.2 gefunden werden.

6 Zusammenfassung und Ausblick

6.1 Ergebnisse

In dieser Arbeit wurde ein Konzept zur Definition von Gebäudemodellvariationen auf Basis von IFC-Daten entwickelt. Das Konzept erlaubt die Speicherung und Generierung von ähnlichen Modellinstanzen aus einem Ausgangsgebäudemodell in IFC. Die für die Modellinstanzen notwendigen Daten werden in einem Variationsmodell hinterlegt, welches durch die Modellierungssprache EXPRESS beschrieben (s. Anhang A.1) und als Textdatei im STEP-Format ausgetauscht wird. Der verwendete Ansatz ist dabei generalisierbar (vgl. Abschnitt 5.2.3) und kann daher auch für andere Modelle genutzt werden. Parametervariationen bilden die Grundbausteine des Variationsmodells und bestehen aus einem Zeiger und den Variationswerten. Der Zeiger referenziert das zu variierende Attribut eines Objektes im Gebäudemodell und wird in EXPRESS mit der Klasse *IfcAttributeBinding* umgesetzt (s. Abschnitt 4.3.1). Die Variationswerte der Parametervariation können hingegen auf zwei Arten dargestellt werden (s. Abschnitt 4.3.2). Zum einen als eine einfache Werteliste (*IfcAttributeVariance*), welche die Variationswerte enthält, oder als Ergebnis einer Operation zwischen zwei Wertelisten (*IfcVarianceResult*). Als Operanden können neben der Werteliste ebenfalls Ergebnisse anderer Mengenoperation verwendet werden, wodurch sich die Variationswerte verschachteln lassen. Als Operationen werden die Mengenoperationen Vereinigung, Durchschnitt und Differenz für das Variationsmodell bereitgestellt. Die folgende Abbildung 6.1 veranschaulicht die Mengenoperationen.

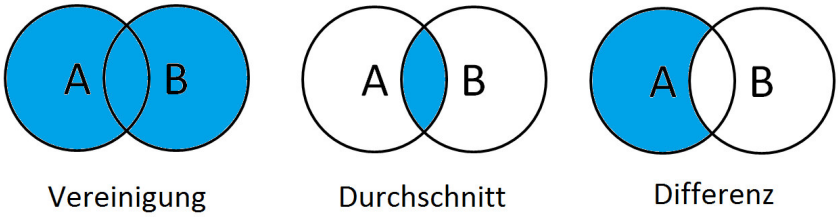


Abbildung 6.1: Mengenoperationen: Vereinigung, Durchschnitt und Differenz

Die dabei existierende Dualität und die damit entstehenden Konflikte zwischen Mengen und Listen werden durch vorgegebene Konventionen verhindert. Während einer Operation werden Wertelisten in Mengen umgewandelt, d.h. die Ordnung der Liste geht verloren und mehrfach auftretende Werte bleiben bis auf einen unberücksichtigt. Die aus der Mengenoperation resultierenden Mengen werden wiederum in Listen umgewandelt, d.h. es wird eine Ordnung für die Wertemenge in aufsteigender Reihenfolge eingeführt.

Um auch Variationen aus mehreren Parametervariationen umzusetzen, lassen sich diese über Verknüpfungsoperationen, die beispielsweise auch Anwendung in SQL und MMQL finden (vgl.

[Fuc15] und [ISOe]), zu Modellvariationen (*IfcModellVariantion*) zusammensetzen. Es werden dabei die drei Verknüpfungsoperatoren INNER JOIN, OUTER JOIN und CROSS unterschieden. INNER JOIN und OUTER JOIN verknüpfen die einzelnen Variationswerte der Parametervariationen nach der Position innerhalb ihrer Variationslisten. Bei INNERJOIN werden unvollständige Wertepaare ausgelassen, während bei OUTERJOIN alle Wertepaare enthalten sind. Die unvollständigen Paare werden dabei mit dem entsprechenden Default-Wert des jeweiligen Parameters ergänzt. Bei der Operation CROSS wird jeder Variationswert einer Parametervariation mit allen anderen Variationswerten der anderen beteiligten Parametervariationen kombiniert. Abbildung 6.2 veranschaulicht die drei Operationen. Detailliertere Ausführungen zu den Verknüpfungsoperationen sind im Abschnitt 4.3.3 zu finden.

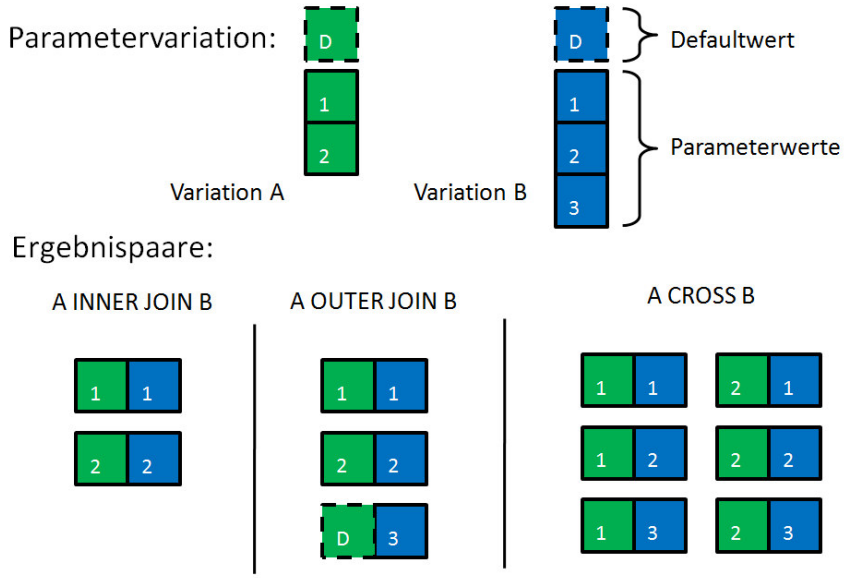


Abbildung 6.2: Verknüpfungsoperatoren: INNERJOIN, OUTERJOIN und CROSS

Mit den vorangegangen Ausführungen lassen sich Modellvarianten eines Gebäudemodells in einem Variationsmodell beschreiben. Das Konzept wurde dazu an einzelnen Beispielen verifiziert (s. dazu auch Abschnitt 5.2.2). Für die Generierung der Modellinstanzen aus dem Variationsmodell wurde zudem eine entsprechende Software-Anwendung entwickelt und an selbigen Beispielen erfolgreich validiert.

6.2 Grenzen und Erweiterungsmöglichkeiten des Variationsmodells

Im Folgenden werden noch bestehende Probleme des Variationsmodells aufgegriffen und untersucht. Zudem wird ein Ausblick für die Weiterführung des bisherigen Konzepts zur Definition von Parametervariation gegeben.

6.2.1 Objektabhängigkeiten von IFC-Gebäudemodellen

Die Validierung der Parametervariation an kleineren Beispielen verlief erfolgreich. Eine Validierung des Konzepts der Parametervariation in IFC konnte jedoch an realen Beispielen noch nicht

vorgenommen werden. Bei komplexeren oder größeren Gebäudemodellen können Schwierigkeiten auftreten, welche durch den hohen Vernetzungsgrad der einzelnen Objekte des Gebäudemodells entstehen. Beispielsweise kann es durch die neuen Parameterbelegungen zu geometrischen Überschneidungen oder Ungenauigkeiten in den Modellinstanzen kommen. Im folgenden Minimalbeispiel soll das Problem aufgezeigt werden. Abbildung 6.3 zeigt das ursprüngliche Gebäudemodell, bestehend aus vier Wänden (*IfcWallStandardCase*), die einen Raum (*IfcSpace*) umschließen.

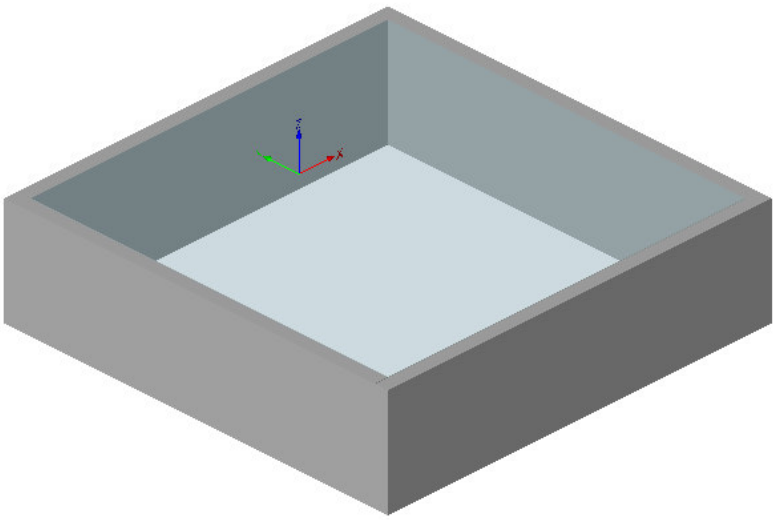


Abbildung 6.3: Raum mit vollständiger Begrenzung durch Wände

Bei einer Variation einer Wand wird dessen Position verändert, wodurch der Raum nicht mehr von den Wänden umschlossen wird (vgl. Abb. 6.4). In verschiedenen Anwendungen, die diese Modellinstanz importieren, können daraufhin Fehlermeldungen kommen, da die Konsistenzbedingung des Raumes nicht mehr erfüllt ist[1].

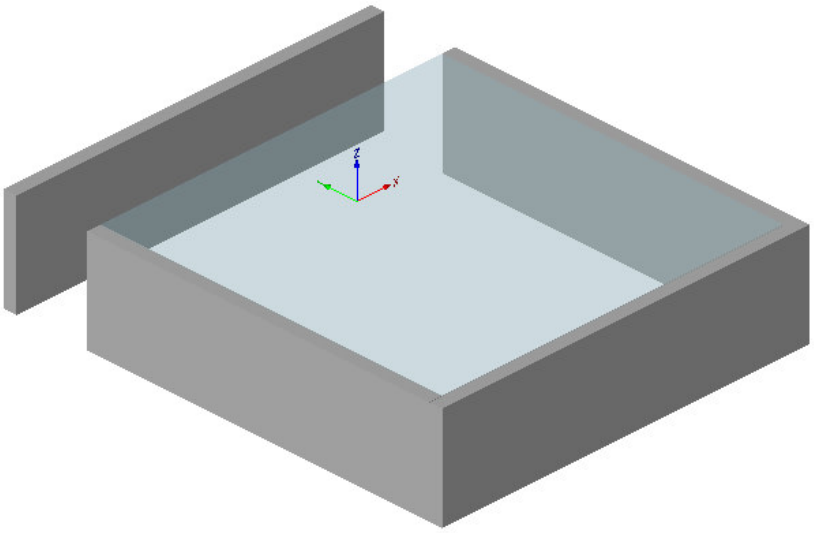


Abbildung 6.4: Raum ohne vollständiger Begrenzung durch Variation

[1]Das Beispiel wurde in der Bauplanungssoftware REVIT importiert, wobei es zu Fehlermeldung kam.

Diese und ähnliche Probleme können nur durch die Berücksichtigung der Abhängigkeiten zwischen den einzelnen Objekten des Gebäudemodells verhindert werden. Grundsätzlich kann das Problem mit dem aktuellen Variationsmodell bereits gelöst werden, indem alle betroffenen Objekte ebenfalls variiert werden. So müssen im oberen Beispiel, die angrenzenden Wände und Räume ebenfalls um das Maß der Verschiebung verlängert werden (vgl. Abb. 6.5).

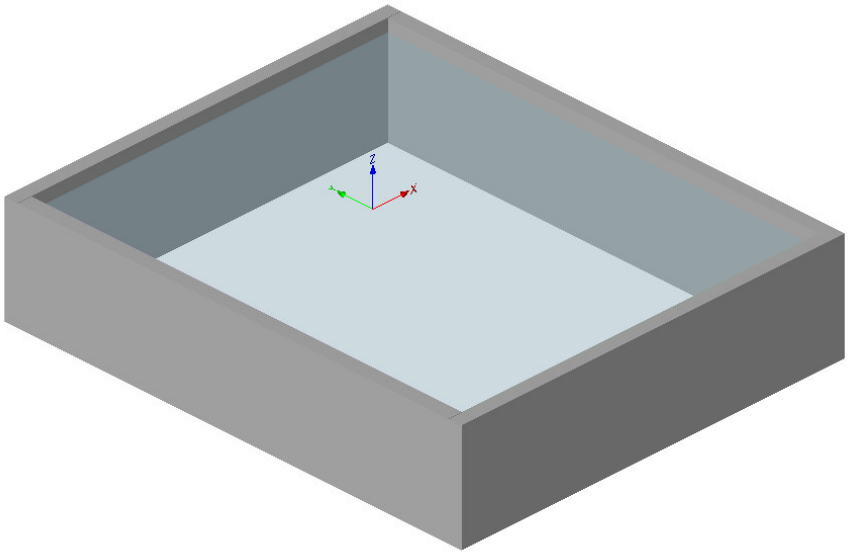


Abbildung 6.5: Raum mit vollständiger Begrenzung durch Variation

Bei kleinen Beipielen wie diesen, kann das händisch geschehen. Bei größeren oder komplexeren Gebäudemodellen muss die Erstellung der Variation computergestützt erfolgen. Abbildung 6.6 zeigt, wie Relationen eines variierten Parameters manuell und computergestützt berücksichtigt werden. In diesem Beispiel wird der Parameter A variiert, was ebenfalls den Parameter B beeinflusst. Bei der manuellen Lösung muss der Anwender die Relation erkennen und entsprechend eine Variation des Parameters B erstellen. Im Falle einer computergestützten Lösung wird dieser Teil automatisiert von einer Software vorgenommen. Die Variation A impliziert folglich die Variation B.

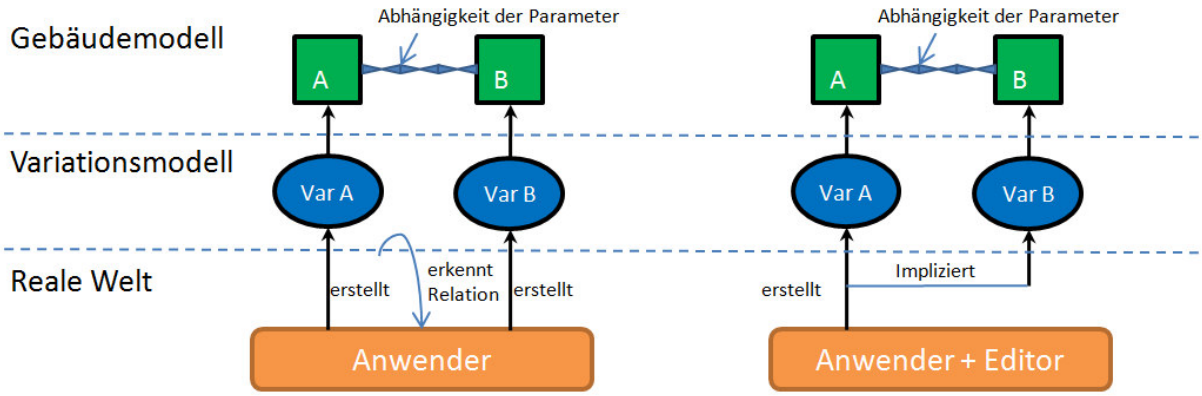


Abbildung 6.6: Berücksichtigung von Parameterabhängigkeiten im Variationsmodell. manuell (links) computergestützt(rechts)

6.2.2 Anwendung zur direkten Erstellung der Variationen

In der Arbeit wurde bereits ein Prototyp des Interpreters für das Konzept der Parametervariation entwickelt und getestet. Dieser erstellt aus dem Gebäudemodell und dem Variationsmodell im STEP-Format die Modellinstanzen mit variierten Parametern. Dadurch ist es bereits möglich, mithilfe des Variationsmodells verschiedene Modellvariationen zwischen den Anwendern auszutauschen und lokal und automatisiert zu generieren. Für die eigentliche Erstellung des Variationsmodells existiert jedoch noch keine Software-Anwendung. Bei einer Fortsetzung dieser Arbeit sollte dies berücksichtigt werden. Die Entwicklung eines Editors zur Erstellung von Parametervariationen ist ein wichtiger Schritt für die Umsetzung und Etablierung in die IFC. Die aktuell noch manuelle Erstellung der Einträge für das Variationsmodell ist sehr zeitintensiv und fehleranfällig. Zudem benötigt der Anwender große Kenntnisse über die IFC und das STEP-Format (s. auch Abschnitt 6.2.1). Mithilfe eines Editors kann die Erstellung des Variationsmodells stark vereinfacht werden. Dazu sollte der entsprechende Editor über folgende Funktionalitäten verfügen:

- **Visualisierung des Gebäudemodells**: Durch die Visualisierung erhält der Anwender einen leichteren Zugang zum Gebäudemodell. Zudem ist die Auswahl des Objektes, an dem die Variation vorgenommen werden soll, stark vereinfacht.

- **EXPRESS Data Browser**: Der Datenbrowser ermöglicht eine einfache Navigation durch die IFC-Datei und zeigt die Beziehungen zwischen den einzelnen Objekten auf.

- **Unterstützte Eingabe von Variationswerte**: Damit der Nutzer nicht alle Variationswerte manuell eingeben muss, sollte eine unterstützte Eingabe bereitgestellt. Beispielsweise können neben der einfachen Listeneingabe auch die Eingabe von Wertebereichen mit Schrittweite angeboten werden. Die Eingabe von allen Zahlen zwischen 1 bis 100 würde beispielsweise mit dem Wertebereich 1-100 und einer Schrittweite von 1 realisiert werden. Computerintern würde der Wertebereich in die vom Variationsmodell geforderte Listenstruktur umgewandelt werden.

- **halbautomatisierte Erstellung des Variationsmodells**: Aus den eingegebenen Parametervariationen soll der Editor eine strukturierte Datei erstellen können, die der Generierung der Modellinstanzen dient. Zudem sollen bei der Erstellung von Parametervariationen relatierte Parameter berücksichtigt und ihre Variationen automatisch ergänzt werden (s. dazu auch Abschnitt 6.2.1).

Die beiden erstgenannten Funktionalitäten besitzen die meisten IFC-Viewer bereits. Abbildung 6.7 zeigt beispielsweise das Anwendungsfenster des Viewers *FZKViewer* des Karlsruher Institute for Technology, welche sowohl eine 3D-Ansicht des Gebäudemodells generiert als auch einen EXPRESS Data Browser besitzt. Die Erstellung des Editors auf Basis eines IFC-Viewers ist damit besonders günstig umzusetzen.

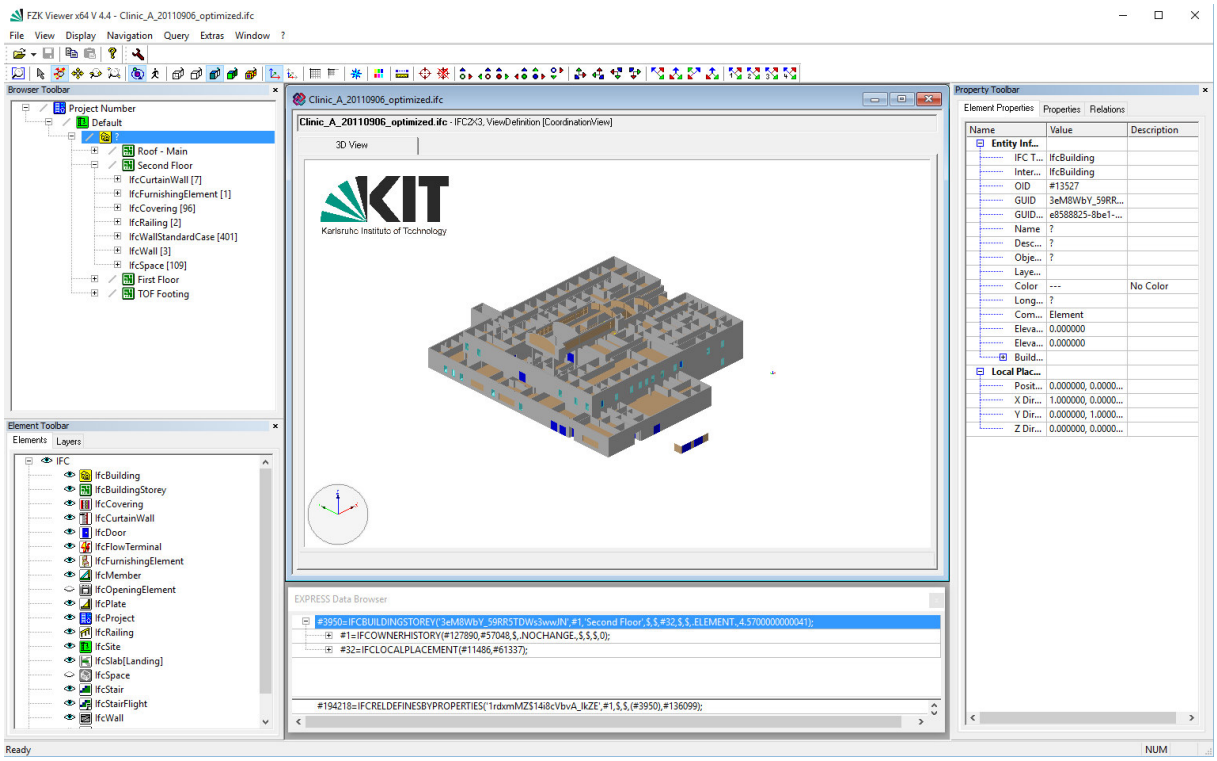


Abbildung 6.7: Anwendungsfenster eines IFC-Viewers

6.3 Fazit

Ein erstes Konzept zur Definition von Gebäudemodellvariationen auf Basis von IFC-Daten konnte in dieser Arbeit bereits gefunden werden. In dem entwickelten Variationsmodell werden die Variationswerte von Parametervariationen integriert und über Zeiger zum IFC-Gebäudemodell mit dem betreffenden Parameter verknüpft. Die Parametervariationen werden im weiteren Verlauf für die Definition von Modellvariation genutzt, welche gegenüber den Parametervariationen eine übergeordnete Einheit darstellen. Sie werden für die Generierung der Modellinstanzen über eine Software ausgewertet. Der entwickelte Ansatz wurde an verschiedenen Beispielen getestet, welche gezeigt haben, dass eine Speicherung und Generierung von Modellvarianten unter Verwendung des erstellten Variationsmodells und Interpreters möglich ist.

Das Variationsmodell berücksichtigt in seiner jetzigen Form noch keine Abhängigkeiten zwischen den Instanzen des Gebäudemodells, weshalb bei der Erstellung einer Variation ihre Abhängigkeiten zurzeit vom Anwender erkannt und die Variation der beteiligten Instanzen des Gebäudemodells noch manuell erstellt werden müssen. Dies stellt eine große Hürde für die Anwendung des Variationsmodells in der Praxis dar. Deswegen sollte, bei einer Fortsetzung dieser Arbeit, die Entwicklung einer Software-Anwendung für eine computergestützte Erstellung von Variationen im Mittelpunkt stehen. Die Anwendung soll dabei die Abhängigkeiten zwischen den Objekten des Gebäudemodells erkennen und automatisch die notwendigen Parametervariationen erstellen.

Literaturverzeichnis

[And92] Anderl, R.: STEP — Grundlagen, Entwurfsprinzipien und Aufbau. 1992, S. 361–381

[AT00] Anderl, Reiner (Hrsg.) ; Trippner, Dietmar (Hrsg.): *STEP STandard for the Exchange of Product Model Data*. Wiesbaden : Vieweg+Teubner Verlag, 2000. – ISBN 978–3–519–06377–3

[Bui] BuildingSMART. *Industry Foundation Classes IFC4 Official Release*. `http://www.buildingsmart.de/`(abgerufen am 27. Juli 2015).

[EHLP13] Egger, Martin ; Hausknecht, Kerstin ; Liebich, Thomas ; Przybylo, Jakob: BIM-Leitfaden für Deutschland. (2013)

[FN13] Fuchs, Sebastian ; Nityantoro, Eko: BIM-Management von Multimodellen. (2013), Nr. SEPTEMBER 2013

[Fuc15] Fuchs, Sebastian: *Erschließung domänenübergreifender Informationsräume mit Multimodellen*. Dresden, 2015. – 241 S. – ISBN 9783867804516

[Gre06] Green, Helen: Building smart. In: *Education* (2006). – ISSN 10809449

[ISOa] ISO 16739:2013: Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries.

[ISOb] ISO DIS 10303-11:2004: Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual.

[ISOc] ISO TC184/SC4: Product data representation and exchange — Part 28: Implementation methods: XML representation of EXPRESS schemas and data.

[ISOd] ISO/CD 10303-21:2002: Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure.

[ISOe] ISO/IEC 9075:2011: Information technology – Database languages – SQL.

[KFB05] Kiviniemi, Arto ; Fischer, Martin ; Bazjanac, Vladimir: Multi-model Environment : Links between Objects in Different Building Models. In: *Proceedings to the 22nd Conference on Information Technology in Construction CIB W78,* (2005)

[LGK14] Lilis, G N. ; Giannakis, G I. ; Kontes, G D.: Semi-automatic thermal simulation model generation from IFC data . Semi-automatic thermal simulation model generation from IFC data. (2014), Nr. August 2015

[Lie09] Liebich, Thomas: buildingSMART IFC2x3 Implementation guide. (2009)

[LK11] Lee, S. H. ; Kim, B. G.: IFC extension for road structures and digital modeling. In: *Procedia Engineering* 14 (2011), S. 1037–1042. – ISSN 18777058

[LT00] Liebich Thomas, Wix J.: IFC Technical Guide. (2000)

[NBY] Nisbet, Nick ; Bonsma, Peter ; Yi, Jang. *Parametrics for IFC (PA-1 Parametric).* `http://www.buildingsmart-tech.org/future/old/ifc-future-extensions/project-proposals/pa-1-parametric` (abgerufen am 6. August 2015).

[Pet07] Petra von Both: *Nemetschek Leitfaden IFC 2x3.* Nemetschek GmbH, 2007

[SA14] Singer, Dominic ; Amann, Julian: Erweiterung von IFC Alignment um Straßenquerschnitte. In: *Proceedings of the 26th Forum Bauinformatik* (2014), S. 1–8

[Sch93] Scheller, Angela: *Informationsmodellierung für verteilte Anwendungen auf Basis standardisierter Datenmodelle.* München : R. Oldenbourg Verlag, 1993. – ISBN 3–486–22754–8

[SS14] Scherer, Raimar J. ; Schapke, Sven-eric: Informationssysteme im Bauwesen 2. (2014), S. 39–64. ISBN 9783662447598

[V.a] o. V. *JFileChooser feste Dateiendung.* `https://www.tutorials.de/threads/jfilechooser-feste-dateiendung.358937/` (abgerufen am 22. Oktober 2015).

[V.b] o. V. *Redirect standard output streams to JTextArea.* `http://www.codejava.net/java-se/swing/redirect-standard-output-streams-to-jtextarea` (abgerufen am 22. Oktober 2015).

[Wix] Wix, Jeffrey. *User Defined Property Sets.* `http://projects.buildingsmartalliance.org/files/?artifact_id=2664` (abgerufen am 10. August 2015).

A Schemata des Variationsmodells

A.1 EXPRESS-Schema des Variationsmodells

```
1  SCHEMA Variationmodell;
2  USE FROM IFC2X4_RC2;
3  ENTITY  IfcModellVariation;
4  JoinMode        :       IfcJoinOperator;
5  Variation       :       SET [1:?] OF IfcVarianceConnector;
6  END_ENTITY;
7
8  ENTITY  IfcVarianceConnector;
9  BoundVariance   :       IfcVarianceOperand;
10 BoundAttribute  :       IfcAttributeBinding;
11 END_ENTITY;
12
13 ENTITY  IfcAttributeBinding;
14 BoundObject     :       IfcObjectSelect;
15 BoundAttribute  :       String;
16 BoundIndex      :       OPTIONAL Integer;
17 END_ENTITY;
18
19 ENTITY  IfcAttributeVariance;
20 Default         :       IfcValue;
21 VarianceList    :       LIST [0:?] OF IfcValue;
22 END_ENTITY;
23
24 ENTITY  IfcVarianceResult;
25 FirstOperand    :       IfcVarianceOperand;
26 SecondOperand   :       IfcVarianceOperand;
27 BooleanOperator :       IfcBooleanOperator;
28 END_ENTITY;
29
30 TYPE IfcJoinOperator = ENUMERATION OF
31 (Cross,
32 InnerJoin,
33 OuterJoin);
```

```
34 END_TYPE;
35
36 TYPE IfcVarianceOperand = SELECT
37 (IfcVarianceResult,
38 IfcAttributeVariance);
39 END_TYPE;
40
41 TYPE IfcObjectSelect = SELECT
42 (IfcClasses);
43 END_TYPE;
44
45 ENTITY  IfcClasses;
46 END_ENTITY;
47
48
49 *TYPE IfcBooleanOperator = ENUMERATION OF
50 *(UNION,
51 *INTERSECTION,
52 *DIFFERENCE);
53 *END_TYPE;
54
55 END_SCHEMA;
```

A.2 XML-Schema des Variationsmodells

```
<?xml version="1.0"?>

<note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.
   w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http
   ://www.w3schools.com note.xsd">

<xs:element name="IfcModellVariation" type="
   IfcModellVariation"  nillable="true"/>
<xs:complexType name="IfcModellVariation">
 <xs:complexContent>
  <xs:sequence>
   <xs:element name="Variation">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="IfcVarianceConnector" minOccurs="1"
         maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute ref="ifc:itemType" fixed="
        IfcVarianceConnector"/>
     <xs:attribute ref="ifc:cType" fixed="set"/>
     <xs:attribute ref="ifc:arraySize" use="optional"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="JoinMode" type="IfcJoinOperator" use="
     optional"/>
 </xs:complexContent>
</xs:complexType>

<xs:element name="IfcVarianceConnector" type="
   IfcVarianceConnector"  nillable="true"/>
<xs:complexType name="IfcVarianceConnector">
 <xs:complexContent>
  <xs:sequence>
   <xs:element name="BoundVariance">
    <xs:complexType>
     <xs:group ref="IfcVarianceOperand"/>
    </xs:complexType>
   </xs:element>
```

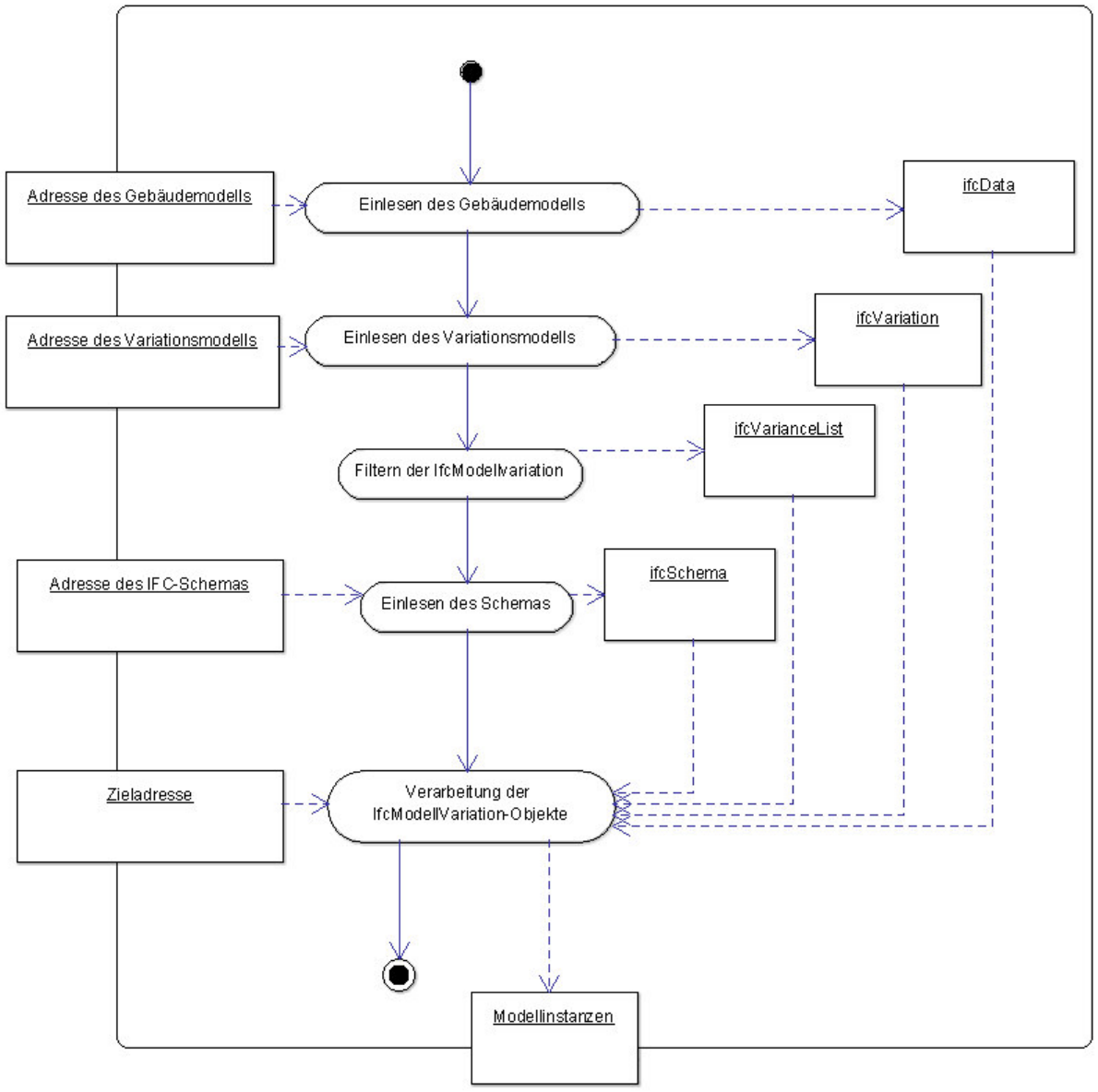
```
33      <xs:element name="BoundAttribute" type="
           IfcAttributeBinding" nillable="true"/>
34    </xs:sequence>
35   </xs:complexContent>
36 </xs:complexType>
37
38 <xs:element name="IfcAttributeBinding" type="
     IfcAttributeBinding"  nillable="true"/>
39 <xs:complexType name="IfcAttributeBinding">
40  <xs:complexContent>
41   <xs:sequence>
42    <xs:element name="BoundObject">
43     <xs:complexType>
44      <xs:group ref="IfcObjectSelect"/>
45     </xs:complexType>
46    </xs:element>
47   </xs:sequence>
48   <xs:attribute name="BoundAttribute" type="xs:string"/>
49   <xs:attribute name="BoundIndex" type="xs:integer" use="
        optional"/>
50  </xs:complexContent>
51 </xs:complexType>
52
53
54 <xs:element name="IfcAttributeVariance" type="
     IfcAttributeVariance"  nillable="true"/>
55 <xs:complexType name="IfcAttributeVariance">
56  <xs:complexContent>
57   <xs:sequence>
58    <xs:element name="Default">
59     <xs:complexType>
60      <xs:group ref="ifc:IfcValue"/>
61     </xs:complexType>
62    </xs:element>
63    <xs:element name="VarianceList">
64     <xs:complexType>
65      <xs:group ref="ifc:IfcValue" minOccurs="1" maxOccurs="
           unbounded"/>
66      <xs:attribute ref="ifc:itemType" fixed="ifc:IfcValue"/>
67      <xs:attribute ref="ifc:cType" fixed="list"/>
68      <xs:attribute ref="ifc:arraySize" use="optional"/>
69     </xs:complexType>
```

```
70      </xs:element>
71    </xs:sequence>
72   </xs:complexContent>
73  </xs:complexType>
74
75  <xs:element name="IfcVarianceResult" type="IfcVarianceResult"
        nillable="true"/>
76  <xs:complexType name="IfcVarianceResult">
77   <xs:complexContent>
78    <xs:sequence>
79     <xs:element name="FirstOperand">
80      <xs:complexType>
81       <xs:group ref="IfcVarianceOperand"/>
82      </xs:complexType>
83     </xs:element>
84     <xs:element name="SecondOperand">
85      <xs:complexType>
86       <xs:group ref="IfcVarianceOperand"/>
87      </xs:complexType>
88     </xs:element>
89    </xs:sequence>
90    <xs:attribute name="BooleanOperator" type="ifc:
        IfcBooleanOperator" use="optional"/>
91   </xs:complexContent>
92  </xs:complexType>
93
94  <xs:simpleType name="IfcJoinOperator">
95   <xs:restriction base="xs:string">
96    <xs:enumeration value="Cross"/>
97    <xs:enumeration value="InnerJoin"/>
98    <xs:enumeration value="OuterJoin"/>
99   </xs:restriction>
100 </xs:simpleType>
101
102 <xs:group name="IfcObjectSelect">
103  <xs:choice>
104   <xs:element ref=""/>
105   <xs:element ref=""/>
106  </xs:choice>
107 </xs:group>
108
109 <xs:group name="IfcVarianceOperand">
```

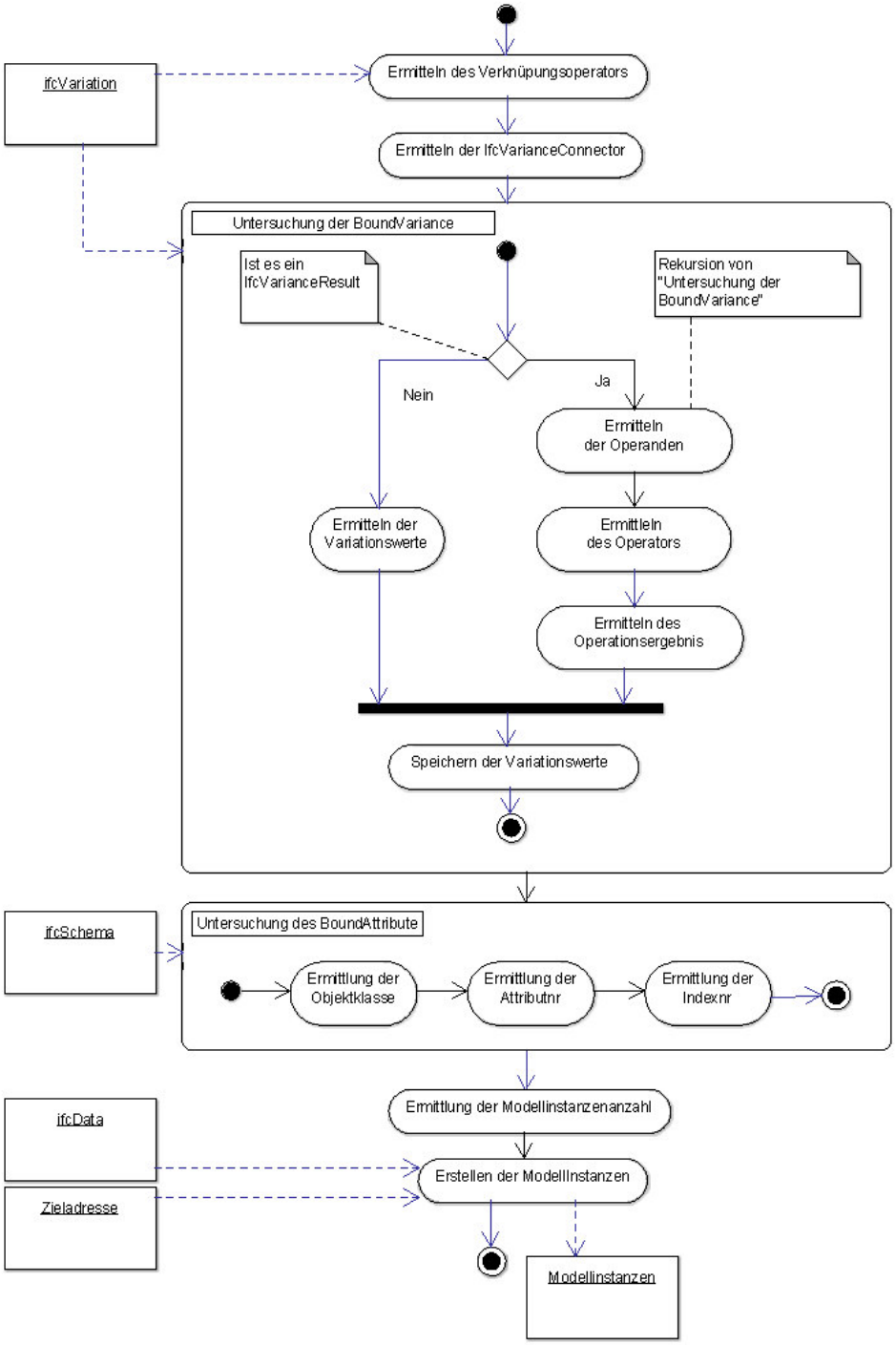
```
110  <xs:choice>
111    <xs:element ref="IfcVarianceResult"/>
112    <xs:element ref="IfcAttributeVariance"/>
113  </xs:choice>
114 </xs:group>
```

B UML-Aktivitätsdiagramm für die Erstellung der Modellinstanzen

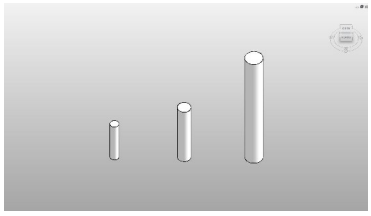
B.1 Gesamte Prozedur



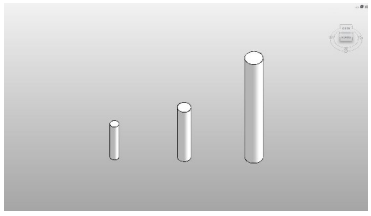
B.2 Teildiagramm: Verarbeitung der IfcModellVariation-Objekte



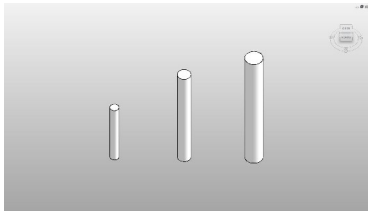
C Visualisierung des Beispiels 3 Stützen



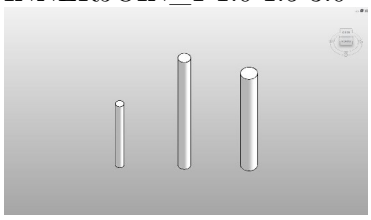
INNERJOIN_1 1.0 1.5 3.0



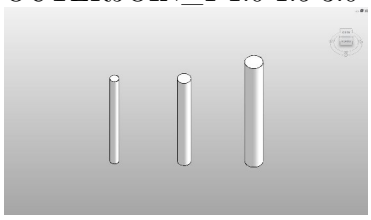
OUTERJOIN_1 1.0 1.5 3.0



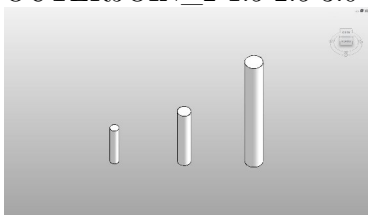
OUTERJOIN_2 1.5 2.5 3.0



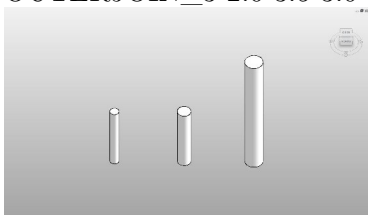
OUTERJOIN_3 2.0 3.5 3.0



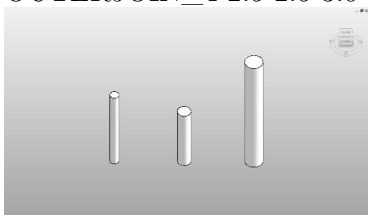
OUTERJOIN_4 2.5 2.5 3.0



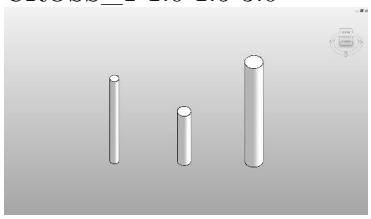
CROSS_1 1.0 1.5 3.0



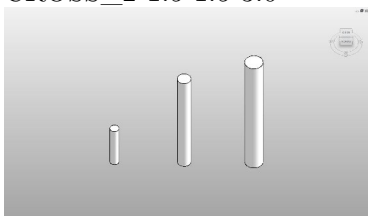
CROSS_2 1.5 1.5 3.0



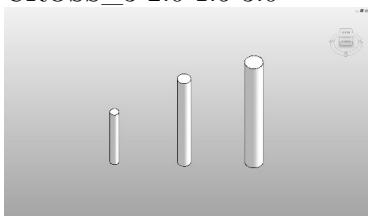
CROSS_3 2.0 1.5 3.0



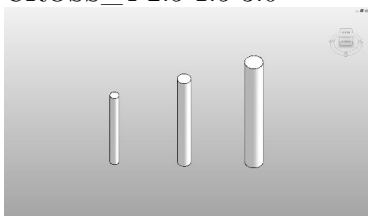
CROSS_4 2.5 1.5 3.0



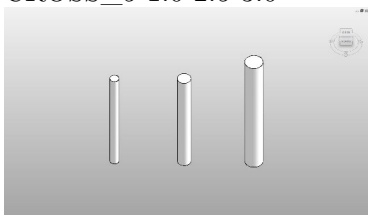
CROSS_5 1.0 2.5 3.0



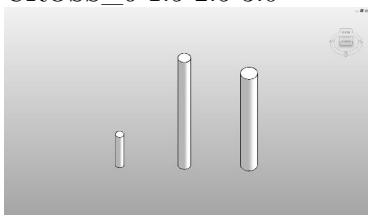
CROSS_6 1.5 2.5 3.0



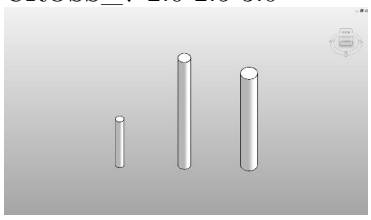
CROSS_7 2.0 2.5 3.0



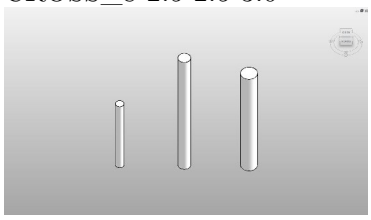
CROSS_8 2.5 2.5 3.0



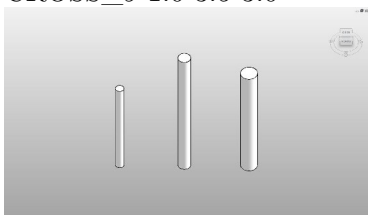
CROSS_9 1.0 3.5 3.0



CROSS_10 1.5 3.5 3.0



CROSS_11 2.0 3.5 3.0



CROSS_12 2.5 3.5 3.0