TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

PROJEKTARBEIT

Evaluierung von Modellprüfungsmethoden auf Basis von IFC

Evaluation of Model Checking Methods
for IFC-based BIM-models

eingereicht von
cand. ing. Lennard Buttgereit
geb. am 04.04.1999 in Berlin
Matrikel-Nummer: 4718286

Verantwortlicher Hochschullehrer / Erstprüfer:

- Prof. Dr.-Ing. habil. Karsten Menzel

Zweitprüfer:

- Prof. Dr.-Ing. Raimar Scherer
- Dr. MD Zabair Sheikh

Wiss. Betreuer/in:

- Emara, Merna, M.Sc.
- Karlapudi, Janakiram, M.Sc.

Mentor IPROconsult GmbH:

- Mischa Sethi, Leiter Digitalisierung und Technologie

Dresden, 7. Juni 2023

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich reiche sie erstmals als Prüfungsleistung ein. Mir ist bekannt, dass ein Betrugsversuch mit der Note „nicht ausreichend“ (5,0) geahndet wird und im Wiederholungsfall zum Ausschluss von der Erbringung weiterer Prüfungsleistungen führen kann.

Name: Buttgereit

Vorname: Lennard

Matrikelnummer: 4718286

Dresden, den 08. Juni 2023

Unterschrift cand. ing. L. Buttgereit

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Aufgabenstellung für das Anwendungsbezogene Wissenschaftsprojekt (BIW5-01)

Name: Lennard Buttgereit (Matrikel Nr.: 4718286)
Vertiefung: Computational Engineering

Thema: **Evaluierung von Modellprüfungsmethoden auf Basis von IFC**
Evaluation of Model Checking Methods for IFC-based BIM-models

Kontext der Projektarbeit:
Die Grundlage für die effektive und zielgerichtete Anwendung des Building Information Modelling liegt in der Vollständigkeit und Korrektheit von Modelldaten. Gewährleistet wird dies durch die Anwendung von Modellprüfungen. Es existieren vielzählige proprietäre, und auch freie Softwarelösungen für diesen Zweck. In der Regel sind dabei Dateien im IFC-Format der Input, welcher anhand vordefinierter Regeln überprüft wird.

Im Zuge des Forschungsprojektes **iECO** wird unter anderem die Übertragung von BIM-Projekten in eine Graphenstruktur untersucht. Die Modelldaten werden üblicherweise aus einer IFC-konformen Datei im ‚step physical file' Format (SPF) in eine Datenbank eingepflegt (entweder objekt-relational oder graphen-basiert).

Zielsetzung der Projektarbeit:
Es ist Ziel der Projektarbeit (i) Modellprüfungsansätze zu klassifizieren und (ii) verschiedene Modelltransformationstools vergleichend zu bewerten. Die Bewertung der Qualität der Übertragung soll sowohl vor als auch nach der Modelltransformation vorgenommen werden.

Klassifizierung von Modellprüfungsmethoden:
Modellprüfungen können unterschiedliche Ziele haben, z.B. geometrische, semantische oder bauregel-basierte Modellprüfungsmethoden.
Prüfung der Modelltransformation als Eingabeprüfung:
Zunächst soll Aufschluss über den Erfolgsgrad der Übertragung der Eingabedaten erhalten werden. Dazu sind IFC-Dateien im SPF als Ausgangsbasis zu nutzen.
Bewertung des Umfangs und der Robustheit bi-lateraler Modelltransformationen:
Tools zur Modelltransformation sollten bi-lateral korrekt und verlustfrei arbeiten. Die Qualität der Modelltransformation aus Datenbanken in SPF-basierte IFC-Dateien ist deshalb in einem zweiten Schritt zu bewerten.

Umfang der Projektarbeit:

Im Rahmen der Projektarbeit sind folgende Fragestellungen zu bearbeiten:

(i) Analyse von am Markt verfügbaren, IFC-kompatiblen Modellprüfungstools, insbesondere der darin enthaltenen Regelsätze.

(ii) Entwurf einer Klassifikation von Modellprüfungsmethoden.
   a. Einschließlich der Beschreibung der Informationsanforderungen für jede Klasse von Modellprüfungsmethoden.

(iii) Entwurf eines Bewertungsschemas zum Vergleich verschiedener Modelltransformationstools.
   a. Modelltransformation von SPF in eine graphenbasierte Datenbank (z.B. Neo4J)
   b. Modelltransformation von SPF in eine objekt-relationale Datenbank (z.B. Oracle)
   c. Modelltransformation zwischen objekt-relationalen und graphenbasierten Informationsmodellen innerhalb einer Datenbank (z.B. Oracle).

(iv) Entwurf, Implementierung und Test eines Modelltransformationstools von SPF-IFC-Dateien in ein IFC-basiertes, objekt-relationales, datenbankgestütztes Informationsmodell.

Verantwortliche und Termine:

Verantwortlicher Hochschullehrer / Erstprüfer:	Prof. Dr.-Ing. habil. Karsten Menzel
Zweitprüfer:	Prof. Dr.-Ing. Raimar Scherer
	Dr. MD Zabair Sheikh
Wiss. Betreuer:	Emara, Merna; MSc.
	Karlapudi, Janakiram; M.Sc.
Mentor IPRO:	Mischa Sethi, Leiter Digitalisierung und Technologie, IPROconsult GmbH
ausgehändigt am:	
einzureichen am:	**16.03.2023**

I ABSTRAKT

Mit der am weitesten verbreiteten IFC-Serialisierung, dem IFC STEP-physical-file, ist es möglich, Bauwerksinformationsmodelle zu speichern und auszutauschen. Dafür wird ein erstelltes Modell aus einer Anwendung in eine Datei exportiert und in eine andere importiert. Der Prozess ist wenig effizient und prädestiniert für redundante Dateien und Informationen. Insbesondere die Abfrage von Dateien nach bestimmten Elementen ist mühsam und erfordert im Regelfall externe Anwendungen. Die nachfolgende Arbeit stellt daher zwei Ansätze für einen dateilosen Informationsaustausch vor, bei denen die Daten persistent in eine graphenbasierte bzw. in eine objektrelationale Datenbank geschrieben werden.

Für den graphenbasierten Ansatz kommt das Labeled-Property-Graph-Prinzip zum Einsatz. Information werden mithilfe von Tripeln, aber auch durch direkte Knoten- und Kanteneigenschaften modelliert. Sogenannte Labels erlauben eine übergeordnete Klassifizierung von Graphelementen. Das dem objektrorientierten Ansatz zugrundeliegende Schema soll eine Relation (Tabelle) für jede konkrete Entität des IFC-Metamodells aufweisen, wobei jedes Attribut der Relation (Spalte) auch ein Attribut der IFC-Entität ist. Dies wird erreicht, indem die Relationen auf der Grundlage strukturierter Typen erstellt werden. Sie sind das objektrelationale Pendant zur objektorientierten Klasse und ermöglichen die objektgebundene Speicherung von Daten. Folglich sind die Tupel einer Relation (Zeilen) tatsächlich Instanzen eines bestimmten strukturierten Typens. Die Bauwerksinformationen werden nach der Überführung aus einer einzigen Quelle heraus manipulierbar und zugänglich durch standardisierte Datenbanksprachen. Als Proof of Concept werden Teile eines Gebäudeinformationsmodells in Form eines IFC-SPF mit einem Python-Algorithmus in die Datenbankmodelle überführt. Die verwendeten Managementsysteme sind Neo4j für die Graphendatenbank und Oracle Database für die objektrelationale Datenbank.

II INHALTSVERZEICHNIS

I Abstrakt I

II Inhaltsverzeichnis II

III Abbildungsverzeichnis IV

IV Tabellenverzeichnis VI

V Abkürzungsverzeichnis VII

VI Quelltextverzeichnis VIII

1 Einführung 1

2 Stand der Technik 3

2.1. Bauwerksinformationsmodellierung (BIM) 3

2.2. Industry Foundation Classes (IFC) 5

2.2.1. EXPRESS Datenmodell 6

2.2.2. Schichtenarchitektur 10

2.2.3. Serialisierung 12

2.3. Dateibasierter Informationsaustausch 14

2.4. Datenbankmodelle 15

2.4.1. Relationale Datenbank 16

2.4.2. Objektorientierte Datenbank 18

2.4.3. Objektrelationale Datenbank 18

2.4.4. Graphendatenbank 21

2.4.5. Weitere No-SQL Modelle 22

2.5. Datenbankbasierter Informationsaustausch 23

2.5.1. BIM Server 23

2.5.2. IfcSQL 24

2.5.3. Objekt-relationaler IFC-Modell Speicher 26

2.5.4. IFC WebServer 28

3 Methodik 30

3.1. Analyse des Modellierungsumfangs 30

3.2. Objektrelationaler Ansatz 34

3.2.1. Objektrelationale Abbildung des IFC-Metamodells 34

3.2.2. Modellierung der Beziehungen 35

3.2.3. Objektorientierte Erweiterungen 39

3.3. Graphenbasierter Ansatz 40

3.4. Verwendete DBMS 41

3.5. Verwendete Programmierschnittstellen 42
3.5.1. Oracle SQL Developer und Data Modeler 42
3.5.2. python-oracledb 44
3.5.3. py2neo 46
3.5.4. IfcOpenShell 49

4 Implementierung 53
4.1. Oracle Objektrelationale Datenbank 53
4.1.1. Datenbankschema 53
4.1.2. Datenbefüllung 63
4.2. Neo4j Graphendatenbank 65

5 Validierung 67

6 Bewertung und Ausblick 71
6.1. Objektrelationaler Ansatz 71
6.2. Graphbasierter Ansatz 73
6.3. Ausblick 74

VII Anlagenverzeichnis XIII

III ABBILDUNGSVERZEICHNIS

Abbildung 1-1: Arbeitsproduktivität je Stunde nach Wirtschaftsbereich 1
Abbildung 1-2: Datenbankbasierter Informationsaustausch 2
Abbildung 2-1: Masterplan BIM 4
Abbildung 2-2: EXPRESS Entität IfcBuilding 7
Abbildung 2-3: Definierter Typ IfcInteger 8
Abbildung 2-4: Entität IfcCartesianPoint 9
Abbildung 2-5: Datenschemata in der Schichtenarchitektur 10
Abbildung 2-6: SPFF für IfcPerson 13
Abbildung 2-7: Objektrelationale Abbildung 19
Abbildung 2-8: Bauwerkstopologie 21
Abbildung 2-9: BIM Server Web-Oberfläche 24
Abbildung 2-10: Pseudocode Struktogramm Wand-Query in IfcSQL 26
Abbildung 2-11: Arbeitsablauf IFC WebServer Graphendatenbank 29
Abbildung 3-1: Teilmodelle CIB-Modell mit a) Architektur, b) Inneneinrichtung, c) Gebäudeautomation und d) TGA 32
Abbildung 3-2: Auszug UML Klassendiagramm Spatial Structure Konzept 36
Abbildung 3-3: Auszug ERM Spatial Structure Konzept – Option 1 36
Abbildung 3-4: Auszug ERM Spatial Structure Konzept – Option 2 37
Abbildung 3-5: Auszug ERM Spatial Structure Konzept – Option 3 38
Abbildung 3-6: Konzepte Objektspalte und Objekttabelle 40
Abbildung 3-7: Graphkonzepte RDF und LPG 41
Abbildung 3-8: Auszug Architektur Oracle SQL Developer Data Modeler 43
Abbildung 3-9: Auszug oracledb Modul 45
Abbildung 3-10: Auszug py2neo Modul 47
Abbildung 3-11: Auszug ifcopenshell Modul 49
Abbildung 4-1: Domain IfcLabel 55

Abbildung 4-2: Strukturierter Typ _IfcRoot und direkte Subtypen 56
Abbildung 4-3: Strukturierte Typen _IfcRoot und _IfcOwnerHistory 57
Abbildung 4-4: Strukturierte Typen _IfcRelContainedInSpatialStructure und _IfcProduct 58
Abbildung 4-5: Kollektionstyp IfcProduct_Collection 58
Abbildung 4-6: Selektionstyp _IfcDefinitionSelect 59
Abbildung 4-7: Entitäten IfcRelContainedInSpatialStructure und IfcPersonAndOrganization 59
Abbildung 4-8: Tabellen IfcRelContainedInSpatialsStructure und IfcOwnerHistory 60
Abbildung 4-9: Spalten der Tabelle IfcOwnerHistory 63
Abbildung 4-10: Systemarchitektur IFC-SPF zu Oracle Database Algorithmus 65
Abbildung 4-11: Arbeitsablauf für Neo4j Graphendatenbank 66
Abbildung 5-1: Oracle SQL Developer – Auszug IfcRelContainedInSpatialStructure 69
Abbildung 5-2: Neo4j Browser – IfcRelContainedInSpatialStructure 70

IV TABELLENVERZEICHNIS

Tabelle 2-1: Auswahl einiger Datenformate verschiedener Domänen 5
Tabelle 2-2: Primitive Datentypen in EXPRESS..... 8
Tabelle 2-3: Aggregierte Datentypen in EXPRESS..... 8
Tabelle 2-4: Datentypen in IFC4..... 9
Tabelle 2-5: Ausgewählte Entitäten des Kernschemas..... 11
Tabelle 2-6: Datenbankbeziehungen 17
Tabelle 2-7: Auszug Datenbankschema IfcSQL 25
Tabelle 2-8: Objektrelationale Tabellenstruktur nach Li et al. (2016)..... 28
Tabelle 2-9: Speicher- und Ladezeiten nach Li et al. (2016) 28
Tabelle 3-1: Datentypen IFC4 Reference View 1.2..... 31
Tabelle 3-2: Anzahl Entitäten des Reference View je Ebene..... 31
Tabelle 3-3: Verschiedene Entitäten und Instanzen in CIB-Modell nach Teilmodell..... 33
Tabelle 3-4: Entitäten und Instanzen im Teilmodell Architektur..... 33
Tabelle 4-1: Abbildung der EXPRESS Datentypen auf Oracle Datentypen..... 54

V ABKÜRZUNGSVERZEICHNIS

AIA	Auftraggeber-Informations-Anforderung
BIM	Bauwerksinformationsmodellierung (en: Building Information Modeling)
CDE	Gemeinsame Datenumgebung (en: Common Data Environment)
DBMS	Datenbankmanagementsystem
DDL	Data Definition Language
IFC	Industry Foundation Classes
ISO	Internationale Organisation für Normung
LPG	Labeled Property Graph
RDF	Resource Definition Framework
SPF	STEP physical file
SPFF	STEP phsical file format
SQL	Structured Query Language
STEP	Standard for the Exchange of Product model data
W3C	World Wide Web Consortium
XML	Erweiterbare Auszeichnungssprache (en: Extensible Markup Language)
XSD	XML Schema Definition

VI QUELLTEXTVERZEICHNIS

Quelltext 2-1: Strukturierter Typ IfcRoot nach Li et al. 27
Quelltext 3-1: Python-Skript – Auszug Datenbankverbindung mit python-oracledb 46
Quelltext 3-2: Python-Skript – Auszug Verbindung zu Neo4j Datenbank mit py2neo 48
Quelltext 3-3: Python-Skript – Parsen eines Modells mit IfcOpenShell 50
Quelltext 3-4: Python-Skript – Parsen einer Instanz mit IfcOpenShell 51
Quelltext 3-5: Python-Skript – Parsen eines Schemas mit IfcOpenShell 51
Quelltext 4-1: SQL-Skript – Typ _IfcRoot 61
Quelltext 4-2: SQL-Skript – Objekttabelle IfcOwnerHistory 61
Quelltext 4-3: SQL-Skript – Enumeration PredefinedType 62
Quelltext 4-4: SQL-Skript – Kollektion IfcProduct_Collection 62
Quelltext 4-5: SQL-Skript – Verschachtelte Tabelle RelatedElements 62
Quelltext 4-6: SQL-Skript - Objekte vom Typ _IfcPerson und _IfcOrganization 64
Quelltext 4-7: SQL-Skript - Objekt vom Typ _IfcPersonAndOrganizationon 64
Quelltext 4-8: SQL-Skript – Objekt vom Typ IfcRelContainedInSpatialContainment 64
Quelltext 5-1: SQL-Skript – Anzahl von Objekten 67
Quelltext 5-2: Cypher-Skript – Anzahl von Knoten 67
Quelltext 5-3: Python-Skript – Anzahl von Instanzen je Entität 68
Quelltext 5-4: IFC-SPF – Auszug Teilmodell Architektur 68
Quelltext 5-5: SQL-Skript – IfcRelContainedInSpatialStructure 69
Quelltext 5-6: Cypher-Skript – IfcRelContainedInSpatialStructure 69
Quelltext 6-1: EXPRESS-Skript – Definition Entität IfcCartesianPoint 72
Quelltext 6-2: EXPRESS-Skript – Definierter Typ IfcLabel 73
Quelltext 6-3: Cypher-Skript – Definition GlobalId Uniqueness und Not Null Constraint 74
Quelltext 6-4: Cypher-Skript – Definition GlobalId Not Null Constraint 74

1 EINFÜHRUNG

Die Bauindustrie ist kaum vergleichbar mit einem anderen produzierenden Gewerbe. Grund dafür ist maßgeblich das Produkt selbst. Jedes Bauwerk, ob Hochbau, Tiefbau oder Infrastruktur ist ein Unikat. Dieses Alleinstellungsmerkmal ist zugleich auch ein Nachteil. Während andere Industriezweige ihre Produktion zunehmend durch Automatisierung optimieren und effizienter gestalten, bleibt das Baugewerbe zurück (siehe Abbildung 1-1).

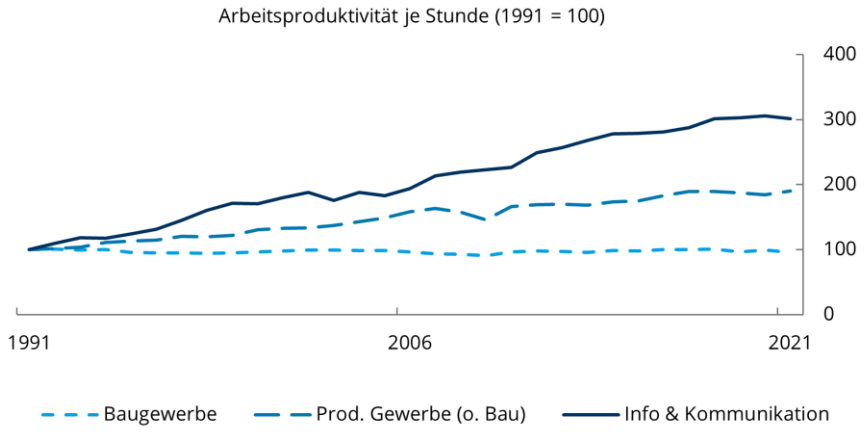


Abbildung 1-1: Arbeitsproduktivität je Stunde nach Wirtschaftsbereich
Datenquelle: Statistisches Bundesamt (2022); eigene Darstellung

Spätestens mit Einführung der Methodik der Bauwerksinformationsmodellierung (BIM) befindet sich das Bauwesen aber auf dem Weg der konsistenten Digitalisierung und damit erhoffter Effizienzsteigerungen. Kern ist das Zusammenwirken möglichst aller am Bau beteiligten Gewerke hin zu einem gemeinsamen digitalen Produkt. Grundlage ist dementsprechend ein ausführlicher Datenaustausch. Zu diesem Zweck existiert das Datenschema der Industry Foundation Classes, kurz IFC. Es wird von der Organisation buildingSMART entwickelt und ermöglicht die nahezu vollständige digitale Beschreibung eines Bauwerks. Die Hauptanwendung des Datenschemas besteht aktuell in der Serialisierung als STEP Physical File, IFC-SPF oder häufig auch IFC-Datei genannt. Der dateibasierte Austausch von Informationen zwischen den Gewerken stellt zwar den aktuellen Standard dar, lässt jedoch noch viel Spielraum für Verbesserungen. Neben redundanten Informationen in Form multipler Dateien, kommt es bei Import und Export von IFC-SPFs aus fachspezifischer Software mitunter zu Informationsverlust (Joseph Jabin et al., 2020). Ohne zusätzliche Applikationen ist es nicht möglich, Daten auf feingranularer Ebene zu extrahieren und maniplieren.

Ein effizienterer Ansatz wäre ein geteiltes Projektmodell mit interoperablem Informationsaustausch. Basis dafür kann ein Datenbanksystem sein, aus dem gezielt Informatio-

nen entnommen und hinzugefügt werden, ohne dafür das gesamte Modell in Dateiform auszutauschen (siehe Abbildung 1-2). Forschungsziel ist daher die Untersuchung verschiedener Datenbankmodelle und derer Anwendungsmöglichkeiten für das IFC-Metamodell. Unter der Vielzahl möglicher Ansätze zeigen sich insbesondere das relationale, objektrelationale und das graphenbasierte Datenmodell als gute Kandidaten. Es werden Anwendungen entwickelt mit dem Ziel, IFC-basierte Bauwerksmodelle in Form eines IFC-SPF In die entsprechenden Datenstrukturen zu überführen. Ergebnis ist eine Grundlage für weitere Untersuchungen, inwiefern sich Datenbanksysteme auch für die anspruchsvolle BIM-Arbeitsweise eignen. Nicht Teil dieser Arbeit ist die tatsächliche Überprüfung der Übertragbarkeit des BIM-Prozesses auf ein datenbankbasiertes Bauwerksmodell.

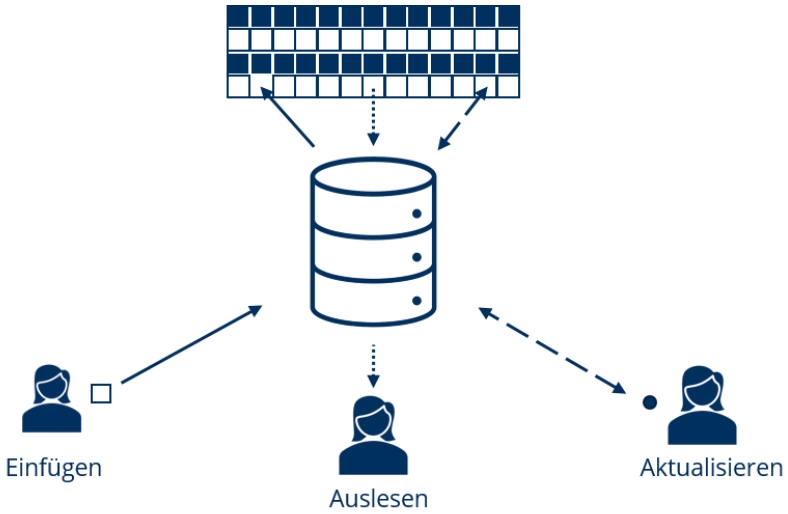


Abbildung 1-2: Datenbankbasierter Informationsaustausch

Ein erster Teil der Arbeit beschäftigt sich mit einer Bestandsaufnahme einerseits der bauwerksorientierten digitalen Modellierung und andererseits dem datenbanktechnologischen Informationsmanagement. Dafür werden die BIM-Methodik und das IFC-Metamodell als zentraler Baustein eingeführt. Datenbankmodelle, sowie ausgewählte bereits untersuchte Ansätze werden vorgestellt. Im nächsten Schritt folgt die Untersuchung, inwiefern die beiden Seiten, die der Bauwerksinformationsmodellierung und die der datenbankbasierten Informationsverwaltung, zusammengeführt werden können. Es kristallisieren sich Ansätze heraus, deren Implementierung Inhalt des dritten Teils ist. Anschließend werden die Implementierungen anhand eines Beispielbauwerkmodells validiert, sowie bewertet und Vorteile bzw. Gegenteile der unterschiedlichen Ansätze herausgearbeitet.

2 STAND DER TECHNIK

Die BIM-Arbeitsweise rund um eine digitale Repräsentation des Bauwerks hat in vielen Bereichen des Baulebenszyklus bereits Einzug erhalten. Während der Erarbeitung, aber auch während der Aufrechterhaltung und Nutzung des Bauwerksinformationsmodells kommen verschiedene bereichsspezifische Anwendungen mit jeweils nativem Format zum Einsatz. Um dennoch einen Informationsaustausch zu gewährleisten, entwickelt die internationale Organisation buildingSMART einen offenen Modellierungsstandard unter dem Namen Industry Foundation Classes (IFC). Das gleichnamige Metamodell kann auf verschiedene Arten serialisiert, also in ein konkretes Bauwerksinformationsmodell überführt werden. Die häufigste ist unter Nutzung eines IFC-STEP-Physical-Files (IFC-SPF), umgangssprachlich auch IFC-Datei genannt. Für den strukturierten Austausch dieser textuellen Repräsentation von Bauwerksinformationsmodellen hat sich weitestgehend der Ansatz einer gemeinsamen Datenumgebung (CDE, en: Common Data Environment) etabliert. Informationen können bis dato jedoch nicht atomar, also elementweise, ausgegeben werden. Ein Konzept, das dazu in der Lage wäre, wird auch als Datenbank bezeichnet. In der Vergangenheit gab es daher Versuche von der dateigebundenen Serialisierung eine datenbankgestützte Form herzuleiten. Nachfolgend werden die Begriffe BIM, IFC und CDE ausführlicher beschrieben, Datenbankmodelle definiert und bereits erfolgte Transformationsversuche vorgestellt.

2.1. BAUWERKSINFORMATIONSMODELLIERUNG (BIM)

BIM, auf Englisch Building Information Modeling, wird nach internationalem Standard definiert als „Nutzung einer untereinander zur Verfügung gestellten digitalen Repräsentation eines Assets zur Unterstützung von Planungs-, Bau- und Betriebsprozessen als zuverlässige Entscheidungsgrundlage“ (DIN, 2019b, S. 13). Die Bauwerksinformationsmodellierung beschreibt im Allgemeinen eine kooperative Arbeitsweise im Bauwesen, bei der ein Bauwerksinformationsmodell als digitale Repräsentation des Bauwerks im Zentrum steht. Bis dato ist unter Projektpartnern allerdings ein Silodenken zu beobachten. Informationsübergabepunkte existieren zwar, beschränken sich jedoch auf ein Mindestmaß an Daten. Darüber hinausgehende, in der Regel bereits digital vorliegende Informationen gehen mitunter verloren. Sie müssen aufwendig von einem anderen Projektpartner erneut erzeugt werden (Borrmann, König et al., 2022). Auch deshalb liegt das Bauwesen bei der Arbeitsproduktivität je Stunde im Vergleich zu anderen Wirtschaftsbereichen zurück (siehe Abbildung 1-1). Genau hier setzt die BIM-Methodik an. Es werden idealerweise sämtliche Informationen untereinander ausgetauscht, zusammengefügt und verwaltet. Ergebnis ist dann nicht eine Ansammlung an Planunterlagen, sondern ein umfassendes digitales Bauwerksinformationsmodell (Borrmann, König et al., 2022). Alle Projektpartner und Gewerke reichern dieses gemeinsame Modell mit Informationen aus

ihrem jeweiligen Fachbereich an, wodurch ein stetiger Informationszuwachs gewährleistet ist.

International gibt es bereits eine Reihe von Ländern, in denen die BIM-Methodik nahezu flächendeckend, mindestens für staatliche Bauvorhaben eingesetzt wird. Vorzuheben sind Großbritannien, Singapur und die USA (Borrmann, König et al., 2022). Spätestens seit dem Masterplan BIM für Bundesbauten steht auch für Deutschland fest, dass die Bauwerksinformationsmodellierung tonangebend wird. Der Masterplan wurde verabschiedet durch das Bundesministerium des Innern, für Bau und Heimat, zusammen mit dem Bundesministerium für Verteidigung und sieht eine stufenweise Einführung der BIM-Methodik für Bundesbauten vor (BMI & BMV, 2021). Nach fast zweijähriger Vorbereitung ist Ende 2022 die erste Stufe verbindlich für alle Bundesbaumaßnahmen geworden. Level Eins beschränkt sich dabei auf die Leistungsphasen 1 bis 5 nach HOAI. Die restlichen Leistungsphasen werden im Zuge der Level Zwei und Drei des Masterplans bis spätestens 2027 abgedeckt (siehe Abbildung 2-1). Es ist zu erwarten, dass durch diese Verbindlichkeit die Umsetzung von BIM in Deutschland weiter an Fahrt aufnimmt.

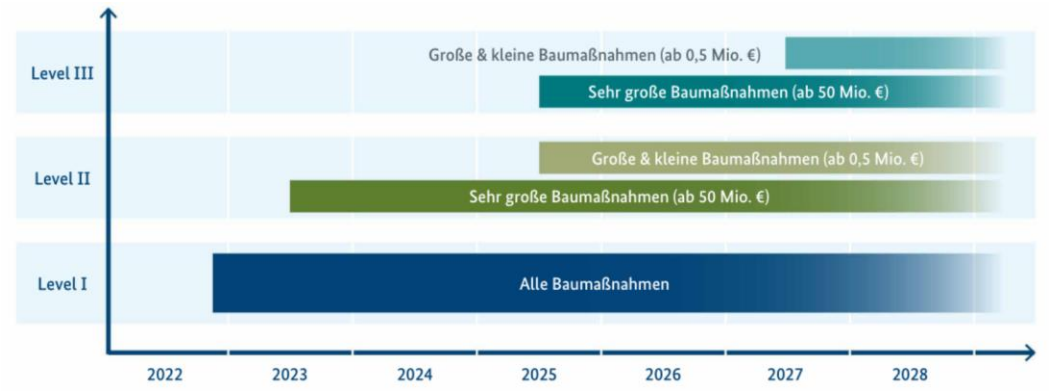


Abbildung 2-1: Masterplan BIM
Quelle: Zielbild BIM für Bundesbauten – verbindliche Einführung von BIM für Bundesbauten (BMI & BMV, 2021)

Um Bauwerksinformationen digital zu ver- und erfassen wird von den verschiedenen am Bau beteiligten Gewerken fachspezifische Software verwendet. Diese unterscheiden sich in ihren nativen Dateiformaten und Beschreibungsansätzen für Bauwerksdaten. Neben diesen nativen oder auch proprietären Formaten haben sich In einigen Bereichen einheitliche Datenformate entwickelt (siehe Tabelle 2-1). Sie ermöglichen den Informationsaustausch innerhalb ihrer Domäne zwischen unterschiedlichen Bearbeitenden, wie auch zwischen unterschiedlicher Software.

Tabelle 2-1: Auswahl einiger Datenformate verschiedener Domänen
Quelle: Fuchs (2015), eigene Darstellung

Datenformat / Bereich	CityGML	CSV	Datanorm	DXF	EnergyPlus	GAEB-DA11	GAEB-D81 bis D86	GLDF	IFC	ISYBAU	LandXML	MS Project	SAF
Abrechnung		✓				✓							
Abwasserplanung										✓			
Ausschreibung							✓						
Baustelle	✓												
Elektro			✓					✓	✓				
Facility Management									✓				
Geodaten	✓										✓		
HLK			✓		✓				✓				
Hochbauplanung	✓			✓					✓				
Bepreisung		✓	✓				✓		✓				
Landschaftsplanung	✓			✓							✓		
Leistungsbeschreibung							✓		✓				
Mengenangabe		✓				✓	✓		✓				
Projektplanung									✓			✓	
Sanitär			✓						✓				
Stadtplanung	✓												
Tragwerksplanung									✓				✓
Vergabe							✓						
Vermessungsdaten											✓		

Der Austausch über die Bereichsgrenzen hinweg ist für die meisten dieser Formate nicht vorgesehen. Zu diesem Zweck existiert jedoch das bereichsübergreifende, neutrale und offene Datenmodell der Industry Foundation Classes (IFC).

2.2. INDUSTRY FOUNDATION CLASSES (IFC)

Bereits zum Ende des vergangenen Jahrhunderts war den Treibern der Bauindustrie bewusst, dass eine fehlende Interoperabilität den Fortschritt der Branche zurückhält. Ohne einen offenen Standard zur digitalen Beschreibung von Bauwerken würde ein intensiver Informationsaustausch unmöglich. Dieser ist jedoch nötig, um Bauprojekte effizienter wie auch weniger fehleranfällig zu gestalten. Zunächst sollte eine Lösung im Rahmen der Internationalen Organisation für Normierung (ISO) erreicht werden. Das

Verfahren im dafür vorgesehenen Standardisierungswerk „Standard für den Austausch von Produktmodelldaten“, kurz STEP, stellte sich jedoch als zu langwierig heraus. Aus diesem Grund wurde 1994 eine eigene Standardisierungsorganisation ins Leben gerufen, die Industry Alliance for Interoperability (IAI), später International Alliance for Interoperabilty. Im Jahr 1997 veröffentlichte sie mit der ersten Version der Industry Foundation Classes (IFC) einen ersten herstellerneutralen, umfassenden Standard zur digitalen Beschreibung von Bauwerken. Die 2005 zu „buildingSMART“ umbenannte Organisation arbeitet seitdem stetig an der Erweiterung und Verbesserung des Schemas. Stand Anfang 2023 ist die aktuelle offizielle Version IFC4.0.2.1 (IFC4 ADD2 TC1). Sie ist rücklaufend auch zum internationalen Standard ISO 16739-1 anerkannt worden (DIN, 2021). Üblicherweise und auch im Rahmen der Arbeit wird sie nachfolgend nur IFC4 genannt. Die chronologisch nächsten erschienen Versionen 4.1 und 4.2 wurden zurückgezogen. IFC4.3 befindet sich im ISO-Verifizierungsprozess und wird bei erfolgreichem Abschluss voraussichtlich IFC4 ablösen (buildingSMART, 2022a). Pläne für ein umfassend aktualisiertes Nachfolgemodell IFC5 bestehen ebenfalls (van Berlo et al., 2021).

2.2.1. EXPRESS DATENMODELL

Konkret sind die Industry Foundation Classes eine Ansammlung von einzelnen Datenschemata, die nach Zusammenführen ein einzelnes computer-interpretierbares Datenmodell ergeben. Ein jedes Schema umfasst dabei digitale Beschreibungen zur Abstraktion unterschiedlicher Teilaspekte des Bauwesens. Darunter fallen baurelevante Objekte mitsamt Attributen und Beziehungen zu anderen Objekten, aber auch Prozesse, Personen und Konzepte wie zum Beispiel eine Kostenanalyse. Das komplexe System Bau wird interpretiert durch ein Zusammenspiel kooperierender Objekte. Als Beschreibungssprache fungiert dabei EXPRESS, eine Datenmodellierungssprache entwickelt im Rahmen des ISO-Standards STEP (DIN, 2021). Die in der Objektorientierung häufig als Klassen bezeichneten Baupläne für Objekte werden in EXPRESS durch Entitäten abgebildet. Im Rahmen der Arbeit werden die Begriffe Klasse und Entität synonym verwendet. An dieser Stelle ist festzuhalten, dass EXPRESS nur dazu dient, das Datenmodell zu beschreiben. Es ist nicht möglich, in EXPRESS konkrete Instanzen einer Entität (Objekte) zu bilden (ISO, 2004). Unter dieser Prämisse können Entitäten im Allgemeinen Attribute, sowie Beziehungen zu anderen Entitäten aufweisen. Grundsätzlich wird eine Beziehung zwischen Entität A und B ausgedrückt, indem Entität A ein Attribut von Typ der Entität B besitzt. Darüber hinaus können Entitäten in Form von Vererbung miteinander in Beziehung stehen. Das Vererbungsprinzip zwischen Entitäten wird in EXPRESS über Super- und Subtypen umgesetzt. Jeder Supertyp gibt die für ihn definierten Attribute und Beziehungen weiter an eine beliebige Anzahl von Subtypen. Das Prinzip der Mehrfachvererbung ist in EXPRESS erlaubt. Jeder Entitätstyp kann demnach mehrere Supertypen haben. Abbildung 2-2 zeigt einen Ausschnitt der EXPRESS-Definition des IFC-Datenmodells, konkret die Definition der Entität IfcBuilding.

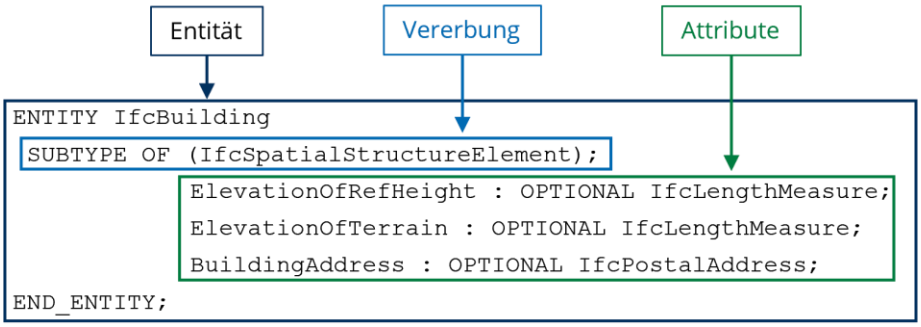


Abbildung 2-2: EXPRESS Entität IfcBuilding

Wie dargestellt ist der Entitätstyp ein Subtyp von IfcSpatialStructureElement. Alle für diesen Typen definierten Eigenschaften gelten auch für IfcBuilding. Folgend sind drei zusätzliche, für diese Entität geltende, Attribute deklariert. Der Beschreibungssyntax lautet Attributname : Attributtyp. Ein zusätzlicher Identifikator OPTIONAL gibt an, dass eine Instanz dieser Entität nicht zwingend einen Wert für das Attribut aufweisen muss. Darüber hinaus ist es möglich explizit inverse Attribute zu deklarieren. Sie stellen keine zusätzliche Information her, ermöglichen aber die beidseitige Navigation zwischen Entitäten. Innerhalb der Definition eines Entitätstypen können durch das Kennwort WHERE Regeln gebildet werden. Deren Einhaltung ist Bedingung für die Gültigkeit einer Instanz dieser Entität. Aktuell bestehen 776 Entitätstypen im IFC-Datenmodell (IFC4). Davon sind 123 Typen abstrakt, was bedeutet, dass von diesen Entitätstypen keine konkreten Instanzen existieren können. Ihr Nutzen liegt in einer verbesserten Struktur und Konsistenz des Datenmodells. Sie definieren Attribute auf einer übergeordneten Ebene für eine Kette von Subtypen. Zum Einen erleichtert dieser Ansatz die nachträgliche Änderung von Eigenschaften für alle Subtypen, zum Anderen wird so eine Informationsredundanz vermieden.

Neben der Entität existieren weitere Datentypen, die der Spezifizierung von Attributen dienen. Die Basis wird durch 7 primitive Datentypen gelegt (siehe Tabelle 2-2). Sie stellen die elementare Ebene dar, können also nicht mit Hilfe anderer Datentypen zusammengesetzt werden.

Tabelle 2-2: Primitive Datentypen in EXPRESS

Datentyp	Beschreibung
Number	Allgemeine Zahl
Real	Reelle Zahl; Spezialfall von *Number*
Integer	Ganzzahl; Spezialfall von *Real*
Logical	*TRUE*, *FALSE* oder *UNKNOWN* mit *FALSE* < *UNKNOWN* < *TRUE*
Boolean	*TRUE* oder *FALSE* mit *FALSE* < *TRUE;* Spezialfall von *Logical*
String	Zeichenkette aus Ziffern, Buchstaben und Sonderzeichen
Binary	Sequenz aus Bits repräsentiert durch 0 bzw. 1

Soll ein Attribut nicht nur einen, sondern mehrere Werte aufweisen, kann dies durch einen der vier vordefinierten Aggregationsdatentypen ausgedrückt werden (siehe Tabelle 2-3). Inhalt einer solchen Sammlung sind Elemente gleichen Datentyps, welcher ebenfalls Teil der Attributbeschreibung ist. Es können eine obere und untere Grenze für die Aggregation deklariert werden.

Tabelle 2-3: Aggregierte Datentypen in EXPRESS

Datentyp	Beschreibung
Array	Geordnete Sammlung fester Größe
List	Folge von Elementen variabler Größe
Bag	Ungeordnete Sammlung variabler Größe; Duplikate erlaubt
Set	Ungeordnete Sammlung variabler Größe; Duplikate *nicht* erlaubt

Neben primitiven Datentypen und Ansammlungen existiert die Möglichkeit eine obligatorische Auswahl als Attributtyp vorzugeben. EXPRESS unterscheidet dabei zwischen Enumeration und Selektionstyp. Ersteres ist die Auswahl aus einer Menge an vorzugebenden Namen. Eine alternative Bezeichnung für diesen Datentyp lautet daher auch Aufzählung. Für den Selektionstyp wird hingegen eine Liste an selbstdefinierten komplexen Datentypen zusammengestellt. Dazu zählen der bereits erläuterte Entitätstyp, sowie eigens definierte Datentypen. Diese können als eine Kombination eines oder mehrerer bereits existierender Datentypen dargestellt werden (Vgl. Abbildung 2-3).

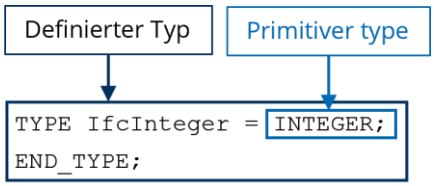


Abbildung 2-3: Definierter Typ IfcInteger

Der Grund für das Neudefinieren eines primitiven Datentyps liegt unter anderem in den Anforderungen an den Selektionstypen. In der Selektionsliste dürfen wie oben beschrieben nur selbstdefinierte komplexe Datentypen vorkommen. Um dennoch beispielsweise eine Ganzzahl einzubinden, kommt eine Hilfskonstruktion wie IfcInteger zum Tragen. Die Kombination der vorgestellten Datentypen ermöglicht beispielsweise die Definition der Koordinaten eines kartesischen Punktes als eine Liste mit 1 bis 3 Elementen vom selbst definierten Typ IfcLengthMeasure, welcher wiederum den primitiven Datentypen Real abbildet (siehe Abbildung 2-4).

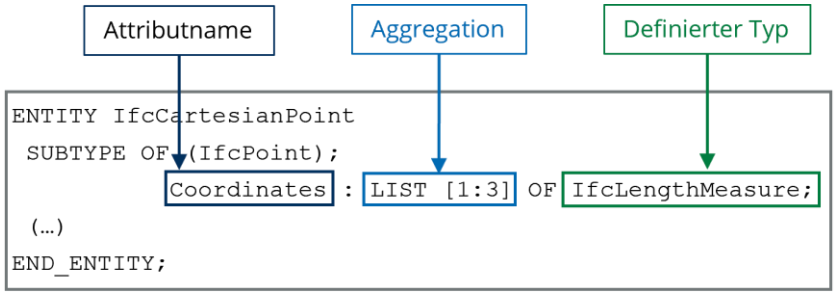


Abbildung 2-4: Entität IfcCartesianPoint

Das Datenmodell der Industry Foundation (IFC4) enthält insgesamt 1173 Entitäten, Attributtypen, Enumerationen und Selektionstypen (vgl. Tabelle 2-4). Von der EXPRESS-Notation abgeleitet existiert eine IFC-Version in der XML Schema Definition (XSD). Diese folgt der im STEP-Protokoll definierten Abbildung von EXPRESS auf XSD (ISO, 2007). Darüber hinaus existiert noch eine offizielle Version in der Web Ontology Language (OWL), die im Kontext des Semantic Web Anwendung findet (W3C, 2023), deren Abbildung allerdings nicht standardisiert ist. Im Rahmen der Arbeit wird ausschließlich die ursprüngliche Definition in EXPRESS behandelt, da sie die Grundlage für alle anderen Versionen bildet.

Tabelle 2-4: Datentypen in IFC4

Datentyp	Anzahl
Entität	776
davon abstrakt	123
Definierter Typ	130
Enumeration	207
Selektion	60

Sowohl EXPRESS als auch die XML Schema Definition und OWL sind nur zur allgemeinen Beschreibung eines Datenmodells gedacht. Das Abbilden der Modelldefinitionen auf konkrete Instanzen und ein konkretes Modell wird auch Serialisierung genannt (siehe Kapitel 2.2.3).

2.2.2. SCHICHTENARCHITEKTUR

Aufgrund der 776 verschiedenen Entitäten und weiteren in Planung wird das IFC-Modell für seine Komplexität teilweise kritisiert. Zur Anwendung des Modells ist es allerdings häufig nicht nötig, die Gesamtheit an Entitäten und Beziehungen untereinander zu kennen. Der Aufbau ist durch eine logische Schichtenarchitektur konsistent und übersichtlich gestaltet (siehe Abbildung 2-5). Für die Fachbereiche des Bauwesens genügt es, bestimmte Teilbereiche des Modells zu verstehen.

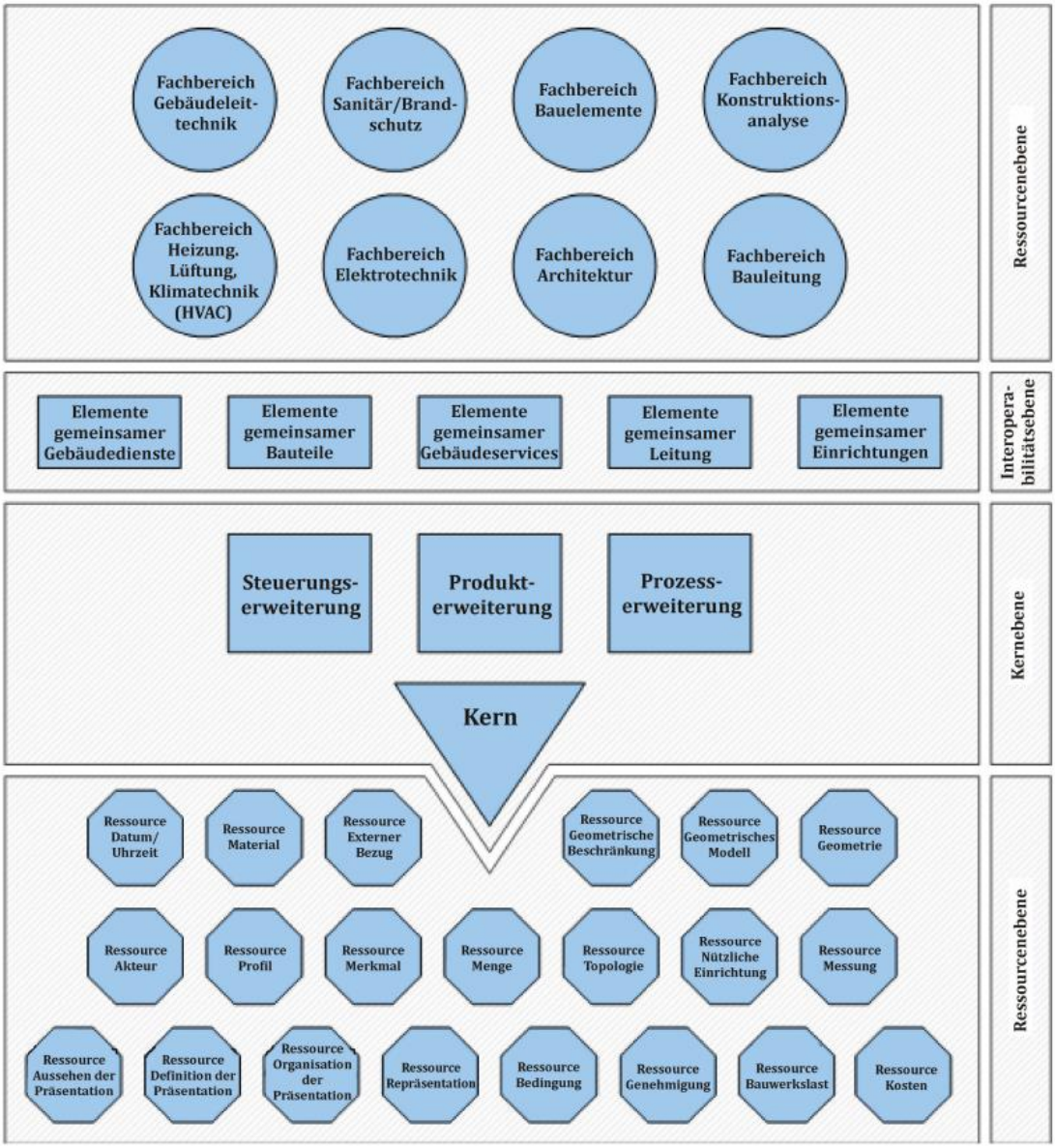


Abbildung 2-5: Datenschemata in der Schichtenarchitektur
Quelle: DIN, 2021, S. 7

Das IFC-Datenmodell ist in vier Schichten respektive Ebenen unterteilt: Kernebene, Interoperabilitätsebene, Fachbereichsebene und Ressourcenebene. Die Unterteilung ist charakterisiert durch einen übergeordneten Anwendungsfall der zugordneten Definitionen auf jeder Ebene. Statt einer losen Ansammlung, werden dabei mehrere Definitionen (Entitäten, Attributtypen, usw.) nochmal in Schemata zusammengefasst.

Die **Kernebene** liefert die Grundstruktur für alle weiteren aspektspezifischen Schemata. Das Kernschema IfcKernel stellt den abstraktesten Teil des Modells dar. Es erfasst allgemeine Konstrukte mit unterschiedlicher semantischer Bedeutung zum allgemeinen Verständnis des Objektmodells, namentlich Objekte, Beziehungen und Eigenschaften. Dazu zählt auch die generelle Basisklasse IfcRoot. Sie ist die Superklasse aller aspektspezifischen Entitäten des IFC-Modells. Durch sie werden Objekte instanziierbarer Subkassen zu folgenden drei Eigenschaften befähigt:

1. Identifizierung - Zuweisung einer global eindeutigen Kennung (GUID – en: Global Unique Identifier)
2. Optionale Eigentümerschaft und Änderungshistorie
3. Optionale Zuweisung von Namen und Beschreibung

Davon abgeleitet sind viele weitere wichtige Basisklassen im Kernschema verortet (siehe Tabelle 2-5). Sie legen zumeist Grundeigenschaften für von ihnen abgeleitete konkrete Klassen in anderen Schemata und Schichten fest.

Tabelle 2-5: Ausgewählte Entitäten des Kernschemas

Name	Beschreibung
IfcActor	An einem Projekt beteiligte Akteure oder menschliche Vertreter
IfcObject	Semantisch behandelte Sache oder Vorgang
IfcProduct	Objekt mit geometrischem oder räumlichem Kontext
IfcProject	Entwurfs-, Konstruktions-, Bau- oder Wartungsarbeit
IfcRelationship	Objektifizierte Beziehung

Die auf dem Kernschema aufbauenden Schemata zur Steuerungs-, Produkt- und Prozesserweiterung dienen der spezifischeren Definition des jeweiligen Bereiches.

Durch die folgende **Interoperabilitätsebene** wird die Verbindung zwischen dem grundlegenden Datenmodellkern und domänenspezifischen Schemata hergestellt. Ein auf dieser Ebene definiertes Schema baut auf den Definitionen der Kernebene auf und wird in der Regel von mehreren Schemata der Fachbereichsebene verwendet.

Die **Fachbereichsebene** enthält abschließende Spezialisierungen. Sie stellen keine Verallgemeinerung darunterliegender Schemata dar. Stattdessen werden in den Schemata dieser Ebene Definitionen für domänenspezifische Objekte, Eigenschaften, Beziehungen

oder Sachverhalte gegeben. Zu den Fachbereichen zählen Gebäudetechnik, Sanitär, HLK, Elektro, Architektur, Bauleitung, Baustatik und Bauelemente.

Auf der **Ressourcenebene** definierte Entitäten können nicht einzeln existieren. Sie müssen von Objekten aus den darüberliegenden Ebenen referenziert werden. Die hier aufgeführten Schemata werden daher auch als unterstützende Datenstrukturen bezeichnet (buildingSMART, 2020a). Keine der Entitäten ist eine Subklasse von IfcRoot. Dementsprechend besitzen Instanzen aus der Ressourcenebene auch keine der oben aufgeführten Eigenschaften wie die global eindeutige Kennung (GUID). Zu Ressourcen zählen die Entitäten, die unter anderem folgende Sachverhalte näher beschreiben:

- Akteur
- Datum & Uhrzeit
- Eigenschaften
- Geometrie
- Kosten
- Material
- Menge
- Topologie

Hier wird das Ineinandergreifen der Kernebene und der Ressourcenebene durch das Kernschema deutlich (siehe Abbildung 2-5). Die Entität IfcActor des Kernschemas bezieht sich beispielsweise direkt auf Entitäten der Ressource Akteur, darunter IfcPerson und IfcOrganization (Person und Organisation).

2.2.3. SERIALISIERUNG

Die Beschreibung eines konkreten Gebäudemodells ist mit EXPRESS nicht möglich. Stattdessen existieren unterschiedliche Serialisierungsmöglichkeiten, die den Definitionen des EXPRESS-Datenmodells folgen und diese auf konkrete Instanzen anwenden. Die meistverwendete ist unter Nutzung des STEP Physical File Formats (SPFF). Umgangssprachlich auch IFC-Datei genannt, handelt es sich bei einem STEP Physical File (SPF) um eine textuelle Repräsentation von Objekten, deren Eigenschaften sowie den Beziehungen unter den Objekten. Die Kodierung ist in der ISO 10303 Teil 21 festgeschrieben (ISO, 2016a).

Jede Instanz einer Entität wird eingeführt mit einer modellinternen eindeutigen Kennung. Sie besteht aus dem Rautezeichen „#“, gefolgt von einer Zahlenkette aus beliebigen positiven Zahlen, mindestens aber einer Zahl ungleich Null (siehe Abbildung 2-6). Über die Kennung kann jede Instanz eindeutig, auch von anderen Instanzen, referenziert werden. Nach diesem Schlüssel kommt die Angabe der Entität. Folgend werden in Klammern stehend, durch Kommata getrennt, Attributwerte zugewiesen. Attributnamen sind nicht Bestandteil der Instanziierung. Eine Zuordnung und Interpretation erfolgt stattdessen durch die Reihenfolge der Attribute. Die Positionierung entspricht der Ordnung, in der die Attribute in dem EXPRESS-Datenmodell definiert wurden. Diese Art der Kodierung erfordert, dass jedem Attribut explizit ein Wert zugeordnet wird. Bei Weglassen von Attributen würde die Zuordnung aufgrund der Verschiebung von Stellen schei-

tern. Da Attribute auch optional sein können, also keinen absoluten Wert aufweisen müssen, bedarf es eines Nullwertes. Für das STEP Physical File Format fungiert das Dollarzeichen „$“ als Angabe eines nicht belegten Attributs. Beziehungen unter Entitäten werden durch Angabe der Entität als Attributtyp modelliert (Vgl. Kapitel 2.2.1). Übersetzt in die Serialisierung wird für ein Attribut vom Datentyp Entität die modellinterne Kennung einer Instanz dieser Entität angegeben.

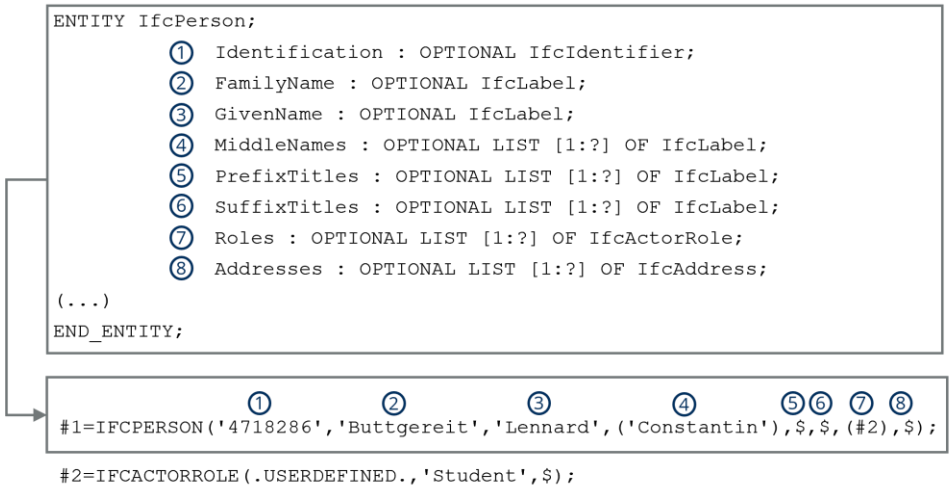


Abbildung 2-6: SPFF für IfcPerson

Neben dem EXPRESS-Datenmodell der Industry Foundation Classes besteht das abgeleitete Modell in der XML Schema Definition (XSD). Analog zum SPFF für EXPRESS, ist dabei die erweiterbare Auszeichnungssprache XML (en: Extensible Markup Language) die Serialisierungsmöglichkeit für das XSD-Datenmodell (W3C, 2023). XML ebenfalls eine textuelle Repräsentation von Objekten, deren Eigenschaften sowie den Beziehungen unter den Objekten. Es hat den Vorteil, dass dessen Kodierung bzw. Syntax auch in anderem Kontext Anwendung findet und damit relativ gängig ist. Zur Speicherung und zum Austausch von virtuellen 3D-Stadtmodellen wird z.B. häufig das auf XML aufbauende CityGML verwendet (Gröger et al., 2012). Daten, die in der Landentwicklungsbranche verwendet werden, wenden in der Regel den offenen Standard LandXML an (LandXML.org, 2014). Nachteilig ist, dass der XML-Syntax zu bedeutend größerem Speicherbedarf führt. Borrmann, Beetz et al. (2022) beziffern den Umfang um ca. 2-3 mal höher als bei einem IFC-SPF gleichen Inhalts. Grundsätzlich lassen sich jedoch die selben Daten und Informationen über beide Formate modellieren. Trotz der zunehmenden Bedeutung, die XML zugesprochen wird, bleibt aktuell die meistgenutzte Serialisierung die über ein STEP Physical File (Amann et al., 2022) und wird daher auch im Rahmen der Arbeit fokussiert.

Es existiert keine Regelung darüber, was konkret der Inhalt eines IFC-SPF oder einer IFC-XML ist. Da es sich um ein Austauschformat für das Bauwesen handelt, ist der Standardanwendungsfall die Beschreibung eines spezifischen Bauwerks oder von Teilen eines

Bauwerks. Teile bezieht sich hierbei nicht nur auf die örtliche Trennung, sondern auch auf die verschiedenen Teilaspekte eines Bauwerks. Häufig wird die Planungsarbeit auf verschiedene Projektbeteiligte verteilt. Ein Architekturbüro ist vorrangig mit dem grundsätzlichen Erscheinungsbild und der inneren Ausgestaltung beauftragt. Ein Statikbüro beschäftigt sich mit der Tragstruktur und einem mechanischen Bauwerksmodell. Ein Büro für technische Gebäudeausrüstung integriert HLK-, Elektro- und GWS-Systeme. So können viele verschiedene Teilmodelle für ein und das selbe Bauwerk entstehen. Für eine integrale und konsistente Planung ist der Austausch dieser Teilmodelle innerhalb des Projektes, unterhalb der Partner, von großer Bedeutung. Nur so kann gewährleistet werden, dass die Teilmodelle und deren Inhalte aufeinander aufbauen und zusammenpassen.

2.3. DATEIBASIERTER INFORMATIONSAUSTAUSCH

Die im Bauwesen nach wie vor am weitesten verbreitete Art des Informationsaustauschs ist mit Hilfe von Dateien. Der Inhalt der Dateien und damit auch deren Informationsgehalt können unterschiedlicher Natur sein: Gebäudepläne, Anträge, Bauteilinformationen, Kalkulationstabellen oder Ablaufpläne, um nur einige zu nennen. Ohne Kontext haben die enthaltenen Daten keinen Informationsgehalt. Auch aus diesem Grund hat sich mittlerweile etabliert, die digitale Repräsentation des Bauobjekts, ein Bauwerksmodell, in den Mittelpunkt zu stellen. Vorher genannte Dateien können entweder inhaltlich aufgelöst und in das Modell integriert oder mindestens als Datei mit dem Modell verknüpft werden. Der offene Modellierungsstandard IFC erlaubt genau diese Überführung von losen Daten zu einem Informationsmodell. Der Austausch eines zusammengeführten Modells beschränkt sich nichtsdestotrotz in der Regel auf den Austausch eines IFC-SPF, also einer Datei.

Der Vorteil dieser Art des Informationsaustauschs ist, dass er mit sehr einfachen Mitteln durchführbar ist. Mittels portabler Medienspeicher wie CD-ROMs und USB-Sticks ist sogar ein physischer Austausch denkbar. Weitaus verbreiteter ist der Austausch über digitale Netzwerkkanäle wie E-Mail. Bedeutend effizienter und strukturierter ist wiederum das Konzept der gemeinsamen Datenumgebung CDE (en: Common Data Environment). Die ISO 19650-1 definiert den Begriff CDE als „vereinbarte Umgebung für Informationen für ein bestimmtes Projekt oder für ein Asset, um jeden Informationscontainer über einen verwalteten Prozess zu sammeln, zu verwalten und zu verbreiten“ (DIN, 2019b). Damit gibt die Norm zwar keinen klaren Rahmen zu den technischen Anforderungen an eine CDE, stellt jedoch die wichtigsten Funktionen vor. Es geht um einen zentralen digitalen Raum, der das Sammeln, Verwalten, Auswerten und Teilen von Informationen ermöglicht. Alle Projektteilnehmer können Daten aus der CDE abrufen und ihrerseits Ausgabedaten ablegen. Mit dem Begriff ist damit eine Datenmanagement-Lösung, wie auch ein Arbeitsablauf verbunden. Die Vorteile offenbaren sich insbesondere bei dem soge-

nannten „federierten“ Informationsmodell. Der Ansatz dahinter ist, dass nicht mehr ein Modell die gesamtheitliche digitale Repräsentation eines Bauprojektes abbildet, sondern dass dies über mehrere lose miteinander verbundene Teilmodelle geschieht. Jeder Projektpartner kann dabei sein dömanenspezifisches Modell mitsamt den verbundenen Informationen in Form von Dateien in einem Informationscontainer ablegen. Zur allgemeinen Abstimmung und Kollisionsprüfung ist es möglich, auf die Teilmodelle anderer Partner zuzugreifen, ohne deren Inhalt oder Urheberschaft zu verändern (Preidel et al., 2022).

Die meisten der technischen Komponenten ähneln denen eines Filehosting-Diensts. Durch die darüber hinausgehenden Definitionen des strukturierten kooperativen Arbeitsablaufs wird daraus eine CDE für Bauprojekte und das anschließende Management von Bauwerken und Liegenschaften. Der Begriff des Filehosting trifft dennoch auf den aktuellen Stand zu. Eine CDE muss nicht in der Lage sein, Dateien aufzulösen und einzeln enthaltene Elemente auszugeben. Dementsprechend ist auch die Verwaltung von einzelnen Objekten in einem Bauwerksmodell nicht zwangsläufig unterstützt. Nach aktueller technischer Regel (DIN, 2019a) wird dieser Stand auch als CDE Level 2 bezeichnet. Legten CDEs Informationen auf einer so feingranularen Ebene ab, dass sich Austauschprozesse auf einzelne Modellelemente oder -attribute beziehen, würde Level 3 erreicht. Stand 2019 ist dies ein „Zukunftskonzept, das bisher technisch jedoch nur in Teilaspekten umgesetzt wurde“ (DIN, 2019a). Preidel et al. (2022) beschreiben dieses Konzept auch als BIM Server. Je nach Kontext kann mit Server eine in einem Rechennetz bereitgestellte physische Maschine (Hardware) oder aber ein in einem Netzwerk abrufbarer Dienst in Form eines Programms (Software) gemeint sein (Schill & Springer, 2012). Was Preidel et al. meinen, ist die Integration eines Datenbank-Servers bis hin zur vollständigen Umstellung von File-Server auf Datenbank-Server.

2.4. DATENBANKMODELLE

Datenbanken sind eine grundlegende Komponente der modernen Informationstechnologie. Sie dienen dazu, große Mengen von Daten zu organisieren und zu speichern, so dass sie schnell und effizient abgerufen werden können (Saake, 2018). Ein wichtiger Aspekt ist die Fähigkeit, Informationen zu verknüpfen und Beziehungen zwischen verschiedenen Datensätzen herzustellen. Es gibt verschiedene Arten von Datenbanken, die je nach Anwendungszweck und Anforderungen eingesetzt werden. Zu den wichtigsten gehören:

1. Relationale Datenbanken
2. Objektorientierte Datenbanken
3. Graphendatenbanken
4. Schlüssel-Wert-Datenbanken
5. Wide-Column-Datenbanken
6. Dokumentenbasierte Datenbanken

Auf einige davon wird im Nachfolgenden näher eingegangen.

2.4.1. RELATIONALE DATENBANK

Die relationale Datenbank ist das bekannteste und am weitesten verbreitete Datenbankmodell (Saake, 2018). Sie basiert auf in Beziehung gesetzten Relationen, die Sammlungen von Daten in Attributen und Tupeln organisieren. Umgangssprachlich ist häufig von Tabellen, organisiert in Spalten und Zeilen, die Rede. Beim Entwurf der Relationen ist es ratsam, einige Regeln zu beachten, um Datenredundanzen zu vermeiden. Geisler (2014) fasst die drei wichtigsten folgendermaßen zusammen:

1. Jedes in einer Relation enthaltene Attribut ist elementar.
2. Jedes Attribut ist entweder vollständig von einem Schlüssel abhängig oder selbst ein Schlüssel.
3. Jedes Attribut, das nicht selbst Schlüsselattribut ist, muss nichttransitiv vom Primärschlüssel abhängen.

Entspricht eine Relation der ersten Regel und es liegen keine mengenartigen Attribute vor, liegt sie in der ersten Normalform (1NF) vor. Listen oder Aufzählungen sind demnach nicht erlaubt. Stattdessen muss eine flache Relation gebildet werden, bei der jedes Mengenelement ein neues Tupel ergibt. Ein nächster Schritt ist, mögliche Redundanzen zu eliminieren. Dafür wird ein Primärschlüssel gebildet. Er kann aus einem oder mehreren (Schlüssel-)Attributen bestehen und muss eindeutig für ein jedes Tupel über die gesamte Relation sein. Anschließend werden die Abhängigkeiten unter den Attributen analysiert. Solche Attribute, die nicht oder nur partiell von dem identifizierten Primärschlüssel abhängen, können ausgelagert werden in eine eigene Relation. Für diese muss jeweils wieder ein neuer Primärschlüssel existieren. Erfolgt dieses Vorgehen für alle Relationen des Datenbankschemas, liegt es in der zweiten Normalform (2NF) vor. Werden auch transitive Abhängigkeiten und implizite Redundanzen bereinigt, ist die dritte Normalform (3NF) erreicht. In der Praxis werden in der Regel Relationen zwischen zweiter und dritter Normalform präferiert (Geisler, 2014). Die Verknüpfung von Datensätzen zwischen Relation A und B erfolgt über Fremdschlüssel. Dabei erhält beispielsweise die Relation A ein Attribut, in dem ein Primärschlüssel von Relation B gespeichert wird. Es gibt die Beziehungstypen 1:1, 1:n und m:n (siehe Tabelle 2-6).

Tabelle 2-6: Datenbankbeziehungen

Typ	Beschreibung
1:1	Jedem Tupel in Relation A ist genau ein Tupel in Relation B zugeordnet.
1:n	Jedem Tupel in Relation A sind null, eins oder beliebig viele Tupel in Relation B zugeordnet.
n:m	Jedes Tupel in beiden Relationen kann beliebig vielen Tupeln in der anderen Relation zugeordnet sein. Es bedarf einer dritten Relation, die die Zuordnungen erfasst.

Umgangssprachlich liegt eine 1:1-Beziehung vor, wenn jeder Primärschlüssel der Tabelle A in genau einer Zeile der Tabelle B vorkommt. 1:n bedeutet hingegen, dass jeder Primärschlüssel aus Tabelle A in beliebig vielen Zeilen der Tabelle B vorkommen darf und bei n:m ist auch die entgegengesetzte Richtung erlaubt.

Um den Zugriff auf und die Verwaltung von Datenbanken zu erleichtern, sind Datenbankmanagementsysteme (DBMS) als Softwarepakete von verschiedenen Herstellern erhältlich. Sie bieten Benutzern eine organisierte Methode zur Speicherung, Verwaltung, Abfrage und Aktualisierung von Daten in einer Datenbank. Dafür wird die eigentliche Datenbank-Engine, die das Speichern und Verwalten durchführt, gekoppelt mit strukturdefinierenden Datenbank-Schemata und einer Reihe von Schnittstellen, die in der Regel über eine Anwendungsoberfläche zur Verfügung gestellt werden. Zu den wichtigsten Anbietern von relationalen Datenbankmanagementsystemen (RDBMS) gehören Oracle, Microsoft, IBM und SAP. In einem Unternehmensreport für die Credit Suisse AG beziffern Zelnick et al. (2017) die Marktanteile nach Jahresumsatz unter benannten Anbietern auf 42% (Oracle), 23% (Microsoft), 14% (IBM) und 7% (SAP).

Um eine Interoperabilität und Datenmigration zu ermöglichen, ist für das relationale Datenbankmodell historisch eine standardisierte Befehlssprache entstanden (ISO, 2016b). Mit der Structured Query Language (SQL) können Relationen erstellt und mit Daten gefüllt werden. Darüber hinaus ist es möglich, vorhandene Daten zu extrahieren, zu sortieren, zu filtern, zu gruppieren und zu aggregieren, kurz, zu manipulieren. Die Syntax von SQL besteht aus einer Reihe von Schlüsselwörtern und Befehlen. Einige der häufig verwendeten SQL-Befehle sind SELECT, INSERT, UPDATE, DELETE, CREATE, DROP und ALTER. In Anlehnung an die Befehlssprache werden relationale Datenbanken auch als SQL-Datenbanken bezeichnet. Alle anderen Datenbankmodelle werden daher auch unter dem Begriff NoSQL zusammengefasst. NoSQL steht dabei für „not only SQL“, also zu Deutsch „nicht ausschließlich SQL“.

2.4.2. OBJEKTORIENTIERTE DATENBANK

Zu den No-SQL-Datenbanken gehören alle Speichermodelle, die nicht ausschließlich auf Relationen beruhen. Das darunter bekannteste ist das objektorientierte. Statt Informationen tupelweise zu verarbeiten, werden sie, wie der Name impliziert, an Objekte gebunden (Saake, 2018). Entwickelt wurden objektorientierte Datenbanken, um die Lücke der persistenten Datenspeicherung in der objektorientierten Programmierung zu schließen. Der Ansatz kam erstmals Ende der 1980er Jahre auf, konnte sich damals aber nicht gegen die bereits etablierten relationalen DBMS durchsetzen. Das liegt unter anderem daran, dass die erhältlichen objektorientierten Datenbankmanagementsysteme (OODBMS) anfangs sehr heterogen waren, während die relationalen Systeme sich gut 20 Jahre zuvor bereits homogenisierten. Auch zeigte sich, dass die Vorteile des objektorientierten Modells nur schwer mit einigen der bekannten DBMS-Prinzipien vereinbar sind.

Die relationale Datenbank ist insofern vergleichbar, als dass eine konkrete Relation ebenso die Attribuierung eines Tupels vorgibt wie eine Klasse die eines Objektes. Im Gegensatz zu Relationen können Klassen dabei aber nicht nur Attribute, sondern auch Methoden definieren. Ein einzelner Datensatz entsteht durch die Instanziierung eines Objektes entsprechend der Klassendefinition. Es wird an eine im Grunde arbiträre Stelle des persistenten Datenbankspeichers geschrieben und erhält einen intern erzeugten Schlüssel, der von da an als Referenz und Zeiger für dieses Objekt wirkt. Der Zeiger wird auch als Objekt-ID (OID) bezeichnet. Dadurch unterscheidet sich der Ansatz maßgeblich vom relationalen Konzept, bei dem Datensätze nur innerhalb einer Relation als Tupel existieren dürfen. Während Tupel im Sinne der Normalisierung außerdem keine mengenartigen Attributwerte aufweisen sollen, ist die Verwaltung komplexer Objekte ein Kernpunkt von OODBMS. Als Datentyp eines Attributes kommen über die primitiven Datentypen hinaus auch Aggregationen, z.B. eine Liste oder ein Set, sowie andere erstellte Klassen als Datentyp in Frage. Trotz der Flexibilität und aufgezeigten Vorteile, haben OODBMS nie wirklich eine feste Rolle in großen Systemen gespielt. Während RDBMS bereits Schnittstellen, Tools und eine einheitliche Befehlssprache aufweisen konnten, befand sich der objektorientierte Datenbankansatz noch in der Entwicklungsphase. Aktuell werden OODBMS nur vereinzelt kommerziell genutzt. Zu den bekanntesten Systemen gehören Versant Object Database, GemStone/s und Objectivity/DB.

2.4.3. OBJEKTRELATIONALE DATENBANK

Wenngleich die rein objektorientierten DBMS aktuell nicht mit RDBMS konkurrieren können, bedarf es einer Lösung zur Datenverwaltung objektorientierter Informationen. Ein Beispiel hierfür ist moderne Software, die in der Mehrheit objektorientiert programmiert ist. Insbesondere Web-Anwendungen müssen in der Lage sein, nutzerspezifische Informationen abzuspeichern und zu gegebenem Zeitpunkt auszugeben. Ein verbreitetes Prinzip der Datenspeicherung von solchen Anwendungen ist daher die objekt-

relationale Abbildung. Dabei werden objektorientierte Daten in ein rein relationales Datenmodell überführt. In der Praxis sind vor allem drei Ansätze verbreitet (Saake, 2018). Es wird eine Relation pro Vererbungshierarchie, eine Relation pro Klasse oder eine Relation pro konkreter Klasse gebildet (siehe Abbildung 2-7).

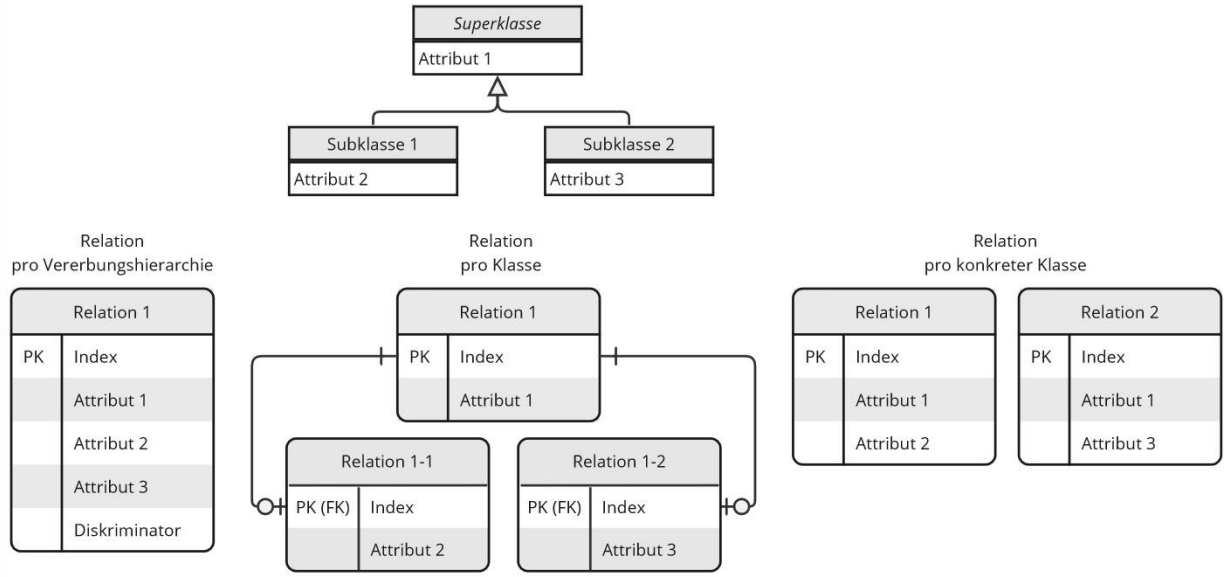


Abbildung 2-7: Objektrelationale Abbildung

Eine Relation für die gesamte Vererbungshierarchie ist auf den ersten Blick sehr übersichtlich. Insbesondere bei großen Hierarchien mit jeweils klassenindividuellen Attributen weist der Ansatz jedoch Schwächen auf. Von einer Vielzahl von möglichen Attributen werden für ein konkretes Objekt nur wenige mit Werten belegt, der Rest ist Null. Hinzu kommt, dass eine Diskriminatorspalte nötig wird, um die tatsächliche Klasse des Objektes anzugeben.

Ansatz zwei bildet daher eine Relation für jede Klasse. Für jedes Objekt muss in der allgemeinsten Superklasse ein Tupel angelegt werden. Über 1-zu-1 Beziehungen zu den Relationen der Subklassen werden die Tupel um klassenspezifische Attribute ergänzt. Konkret wird der Primärschlüssel des Supertypen auch als Primärschlüssel für die Subtpyen genutzt. Dadurch können über die Relation der Basisklasse alle Datensätze der gesamten Hierarchie abgerufen werden. Ein Nachteil dieser Modellierung ist, dass auch abstrakte Klassen als Tabelle abgebildet werden müssen. Ein Tupel einer solchen Relation ist dann nicht direkt als Objekt interpretierbar, sondern nur ein Hilfsdatensatz. Zudem sind für ein einziges Ursprungsobjekt unter Umständen eine Vielzahl von Tupel nötig.

Der dritte Ansatz sieht vor, nur die konkreten Klassen der Vererbungshierarchie auf Relationen abzubilden. Dabei erhält die Relation nicht nur die Attribute der konkreten Klasse, sondern auch die vererbten Attribute der Superklassen. Dadurch ist jedes Tupel

direkt interpretierbar als Objekt der zugeordneten Klasse. Nachteilig ist hier die Gefahr, Informationen redundant zu speichern, da alle Subklassen ggfs. die vererbten Attribute der Superklasse abbilden. Außerdem ist es nicht wie nach Ansatz 2 direkt möglich, alle Datensätze einer Vererbungshierarchie abzufragen.

Der Vollständigkeit halber zu benennen ist das Verfahren der Objektabstraktion. Dieser Ansatz findet kaum noch Anwendung (Saake, 2018). Die Objektstrukturen werden komplett aufgelöst und auf mindestens fünf generalisierte Relationen verteilt:

1. Relation für Klassen
2. Relation für Beziehungen einschließlich Vererbungen
3. Relation für Attribute
4. Relation für Klasseninstanzen
5. Relation für Attributwerte

Als Alternative zur objektrelationalen Abbildung haben viele RDBMS ihr Angebot um objektorientierte Funktionen erweitert. Sie werden dann auch als objektrelationale Datenbankmanagementsysteme (ORDBMS) bezeichnet. Die grundlegende Strukturierung von Informationen folgt dem Konzept der Relationen. Zusätzlich können aber auch Klassen angelegt und Objekte gespeichert werden. Das bringt den Vorteil, dass als Datentyp eines Relationattributs nun die zuvor angelegten Klassen möglich sind. Gespeicherte Objekte werden in einem Tupel mit dem entsprechenden Attribut referenziert. Alternativ ist es auch möglich, ganze Relationen auf den Definitionen einer Klasse aufzubauen. Bei dem Konzept der Objekttabelle werden die Informationen objektorientiert gespeichert und als Relation präsentiert. Jedes Tupel spiegelt im Grunde die Attributwerte eines konkreten Objektes wider.

Eine weitere Erweiterung sind Aggregationsdatentypen. In der objektrelationalen Mischform ist es erlaubt, eigene Kollektionsdatentypen zu definieren. Unter Angabe des Datentyps der enthaltenen Elemente wird entweder ein in Größe beschränkter Array oder eine variable Collection angelegt. Anschließend kann dem Attribut einer Relation oder einer Klasse jener Kollektionsdatentyp zugewiesen werden. Technisch wird die Instanziierung einer Kollektion umgesetzt durch verschachtelte Tabellen. Es wird zusätzlich zur eigentlichen Relation eine weitere Relation angelegt, die nur über das entsprechende Attribut zugänglich ist. Hier sind die einzelnen Elemente des Arrays oder der Collection gespeichert und referenzieren ein bestimmtes Tupel der Ursprungsrelation.

ORDBMS kommen vor allem dann zum Einsatz, wenn objektorientierte Daten mit nicht objektorientierten verknüpft werden sollen. Eine andere Motivation ist die bessere Verwaltung von Objekten mit einer bekannten Strukturierung und Oberfläche. Systeme, die diesen Ansatz zu Teilen oder vollständig unterstützen, sind Oracle Database, IBM DB2 und Microsoft SQL Server. Im Grunde sollte jedes relationale DBMS, das SQL:1999-konform ist, auch die objektorientierten Erweiterungen abbilden können. Hier wurde

erstmals das Klassenprinzip durch sogenannte strukturierte Typen eingeführt (ISO, 1999).

2.4.4. GRAPHENDATENBANK

Ein Graph ist eine Struktur aus Knoten und Kanten, wobei die Menge an Kanten die Menge an Knoten verbindet. Kanten können entweder zwei Knoten miteinander verbinden oder einen Knoten mit sich selbst in Beziehung setzen. Sie können ungerichtet, einseitig oder beidseitig gerichtet sein. Graphen eignen sich „insbesondere für die Darstellung von Beziehungen zwischen Objekten“ (Saake, 2018). Eine solche Beziehung kann beispielsweise die Topologie eines Bauwerks sein (siehe Abbildung 2-8).

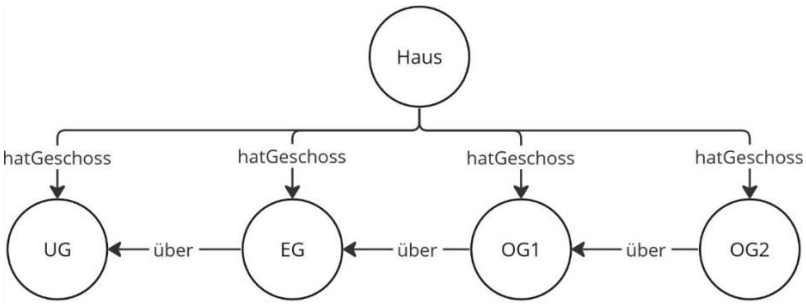


Abbildung 2-8: Bauwerkstopologie

Für Grapheninformationsmodelle existieren grundsätzlich zwei verschiedene Ansätze. Das Resource-Description-Framework (RDF) ist die grundlegendste Möglichkeit, Informationen in Graphen darzustellen. Es handelt sich um einen Definitionsrahmen des World Wide Web Consortiums (W3C), der auf dem Satzbauprinzip basiert (W3C, 2014). Das bedeutet, dass sämtliche Informationen als Subjekt-Prädikat-Objekt Tripel modelliert werden. Subjekt und Objekt sind Knoten, die durch das Prädikat als Kante in Beziehung stehen. Durch den Syntax sind Kanten im RDF grundsätzlich immer gerichtet. Für die Bauwerkstopolgie ist ein solches Tripel „Haus-hatGeschoss-UG“. Dem gegenüber stehen Labeled Property Graphs (LPG). Ein LPG erweitert das Subjekt-Prädikat-Objekt-Modell um Eigenschaften, die Knoten und Kanten direkt zugewiesen werden. Soll im RDF beispielsweise der projektspezifische Name des Hauses ergänzt werden, muss mindestens eine neue Kante und ein neuer Knoten modelliert werden. Denkbar ist ein Tripel „Haus-hatName-NürnbergerEi“. Für ein LPG kann diese Information auch dem Knoten Haus direkt als Wertepaar gegeben werden. Ohne zusätzliche Elemente werden so insbesondere die im RDF Literal genannten primitiven Datentypen modelliert. Hingegen komplexe Datenstrukturen sind weiterhin als Tripel zu modellieren. Es ist nicht ratsam, Geschosse als Eigenschaften für das Haus anzulegen, da diese selbst wieder neue Informationen tragen. Textuell werden die Labeled Properties in der Regel in Klammern hinter einen Knoten geschrieben. Bei der Tripel-Darstellung entsteht folgendes Konstrukt:

„Haus{Name: NürnbergerEi}-hatGeschoss-OG2{Name: CIB}“. Zusätzlich kann auch der Kante eine beliebige Menge an Eigenschaften zugewiesen werden. Hinzu kommen sogenannte Labels als Menge von Typbezeichnungen, die Knoten und Kanten kategorisieren.

Zu den am weitesten verbreiteten Datenbankmanagementsystemen auf Graphenbasis gehören Neo4j, Amazon Neptune und OrientDB. Sie basieren größtenteils auf einem generischen LPG-Ansatz. Das RDF wird hingegen vor allem zur Wissensmodellierung und im Kontext des semantischen Webs verwendet.

2.4.5. WEITERE NO-SQL MODELLE

Das simpelste Informationsmodell bilden **Schlüssel-Wert-Datenbanken**. Der Schlüssel ist eine eindeutige Kennung für einen Datensatz und der Wert der tatsächliche Datensatz. Schlüssel-Wert-Datenbanken sind einfach und flexibel in der Handhabung und eignen sich besonders für die Verarbeitung von großen heterogenen Datenmengen, die schnell und effizient abgerufen werden müssen. Es gibt keine Anforderungen an den Datentypen des Datensatzes, es existiert keine strikte Modellierungsvorgabe und keine Schemabindung. Sie sind daher auch sehr gut skalierbar, egal ob vertikal auf einem Server oder horizontal über mehrere Server verteilt (Vossen, 2008). Bekannte Anbieter sind Redis und BerkeleyDB.

Auf dem Prinzip aufbauend sind **Wide-Column-Datenbanken** entstanden. Auch hier kommen Schlüssel-Wert Paare zum Einsatz, wobei der Wert selbst wieder eine Menge an Schlüssel-Wert-Paaren ist. Letztere sind eine Kombination aus einem Attributnamen als Schlüssel und dem Attributwert als Wert (Geisler, 2014). Es ist daher teilweise auch von zweidimensionalen Schlüssel-Wert-Datenbanken die Rede. Die dadurch entstehende visuelle Struktur ähnelt der eines RDBMS, wobei die Attribute, also Spalten, nicht für alle Datensätze gleich sind. Mit der Speicherstruktur eines RDBMS haben Wide-Column-Datenbanken allerdings nichts zu tun. Der häufig verwendete Begriff spaltenorientierte Datenbank ist daher auch reichlich unglücklich gewählt. Hierunter werden nämlich eigentlich RDBMS zusammengefasst, die die Daten nicht zeilenweise persistent speichern, sondern spaltenweise. Für Wide-Column-Datenbanken existieren unterschiedliche Managementsysteme, darunter Apache Cassandra, Google Bigtable und ScyllaDB.

Ein weiterer Spezialfall der Schlüssel-Wert-Datenbank ist die **dokumentenbasierte Datenbank**. Hierbei ist jeder Wert zu einem Schlüssel ein Dokument in einem maschinenlesbaren Format, häufig XML oder JSON. Ein Dokument kann beliebige Informationen in einer willkürlichen inneren Struktur tragen. Die hohe Flexibilität dieses Ansatzes steht der fehlenden Beziehungsmodellierung gegenüber (Saake, 2018). Das bekannteste DBMS für diese Art von Informationsmodell ist MongoDB.

2.5. DATENBANKBASIERTER INFORMATIONSAUSTAUSCH

Sollen einzelne Elemente eines Bauwerkmodells unmittelbar verändert oder erweitert werden, scheint die dateibasierte, textuelle Repräsentation in Form eines IFC-SPF nur bedingt geeignet. Das IFC-Schema zur digitalen Beschreibung eines Bauwerks ist aber nicht limitiert auf die Serialisierung mithilfe des SPFF oder XML. Stattdessen können auch andere Datenstrukturen verwendet werden. Datenstrukturen, deren Anwendungsbereich prädestiniert ist für die Verwaltung von Daten, aber auch den interoperablen Datenaustausch zwischen Projektpartnern unterstützen. In vielen anderen Bereichen hat sich hierfür das Datenbankkonzept durchgesetzt. Datenbankmanagementsysteme verschiedener Anbieter haben über die letzten Jahrzehnte eindrucksvoll bewiesen, wie sie den üblichen Problemen des dateibasierten Informationsaustausches entgegenwirken. Bereits Anfang der 2000er Jahre wurden Versuche unternommen, sich die Vorteile eines DBMS für das Bauwesen zu Nutze zu machen. Kiviniemi et al. fassen die damaligen Bemühen zusammen und stellen fest, dass zwar einige Projekte vielversprechende akademische Ergebnisse erzielen, i.d.R. aber ein Proof-of-concept für die Industrie fehlt. Grund dafür sehen sie weniger in der Datenstruktur als einer geeigneten Benutzerschnittstelle. Trotz des sich abzeichnenden Potenzials wurde ein Großteil der Projekte leider nicht weiterverfolgt. In der Zwischenzeit sind neue Anwendungen entwickelt worden, von denen einige nachfolgend vorgestellt werden.

2.5.1. BIM SERVER

Das vielleicht vielversprechendste Projekt läuft unter dem Namen BIM Server, anfänglich IFC Server (Beetz et al., 2010). BIM Server versteht sich als IFC-basierter Open-Source-Bauwerksiformationsmodell Server, der die Datenverwaltung mit einer Softwareentwicklungsplattform vereint. Darauf aufbauend sind in der Vergangenheit Anwendungen wie z.B. ein Model Viewer entstanden (Krijnen et al., 2023). Als Basistechnologie kommt die von Oracle gekaufte Open-Source Lösung Berkeley DB zum Einsatz. Es handelt sich um eine „transaktionale Schlüssel-Wert-Datenbank-Engine, entworfen, um der Anwendungsentwicklung die beste Zugriffsinfrastruktur auf eine Kern-Engine zu ermöglichen, die den Bedürfnissen der Anwendung entspricht“ (Yadava, 2007).

BIM Server hat um die No-SQL-Datenbank herum eine Applikation entwickelt, die es Nutzern auf einfache Weise möglich machen sollte, IFC-basierte Programme zu schreiben. Es handelt sich nicht um eine relationale Datenbank oder ein relationales Datenbankmanagementsystem. Daher wird auch nicht die standardisierte Abfragesprache SQL unterstützt. Stattdessen sind Operationen mit Datensätzen, also Erstellung, Ausgabe, Veränderung und Löschung, als Module programmiert und in die Plattform integriert. Ausführbar sind sie über die plattforminterne Konsole und in einigen Fällen auch über eine Benutzeroberfläche (siehe Abbildung 2-9).

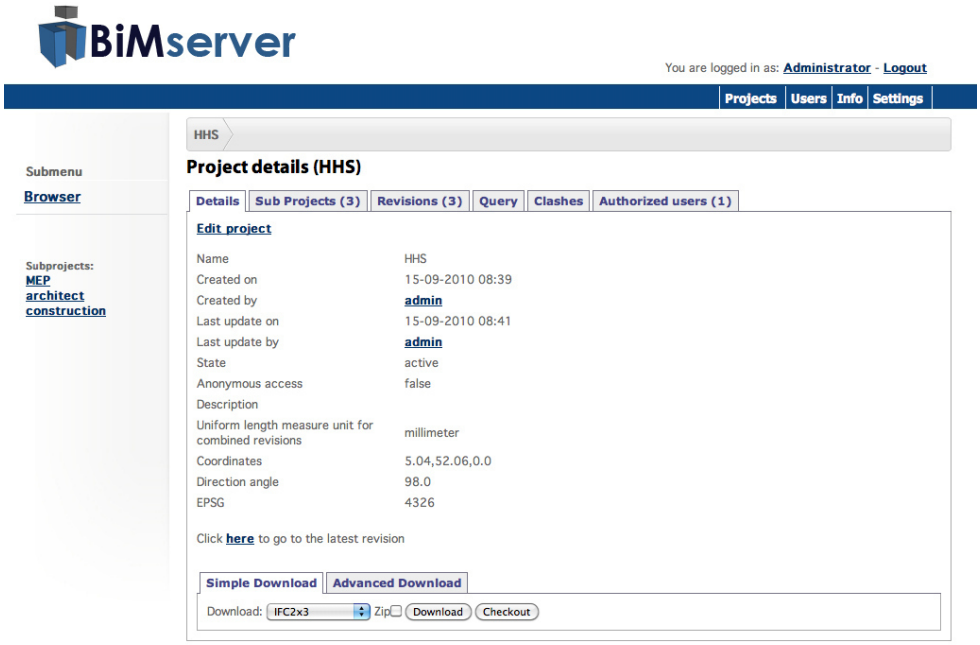


Abbildung 2-9: BIM Server Web-Oberfläche
Quelle: Krijnen et al. (2023)

Im Fall des BIM Server ist auf Grundlage der Berkeley DB eine frei zugängliche Plattform zur BIM-Anwendungsentwicklung entstanden. Es können Projekte und Nutzer erstellt, Zugriffsrechte verteilt und Informationsmodelle erstellt werden. Die Hauptfunktion besteht in der Transformation eines IFC-SPF in die Struktur der Datenbank. Es ist möglich, anders als beim dateibasierten Austausch, konkret einzelne Objekte eines Bauwerksinformationsmodell auszugeben. Da der Fokus jedoch nicht auf der Datenverwaltung liegt, ist die Abfrageform weder trivial, noch leicht zugänglich. Der Anwenderin und dem Anwender wird eine begrenzte, wenn auch umfängliche Auswahl an Datensatzmanipulationen an die Hand gegeben, deren Dokumentation leider nicht vollständig ist. Stand Anfang 2023 ist der BIM Server weiterhin als Open-Source-Lösung verfügbar, wird jedoch seit 2015 nur noch gewartet und zur Verwendung in aktuellen Anwendungsumgebungen angepasst (Krijnen et al., 2023).

2.5.2. IFCSQL

Aufbauend auf den Erkenntnissen aus Projekten wie dem BIM Server, sind andere Datenstrukturen als Basis für die Verwaltung von Bauwerksinformationsmodellen untersucht worden. Mit einem Fokus weniger auf die Anwendungsentwicklung und mehr auf das tatsächliche Datenmanagement, setzen Bock und Eder (2022) auf einen relationalen Datenbankansatz. Sie liefern mit einem Open-Source Quellcode das Fundament, um ein Microsoft SQL Server 2019 Datenbankschema aufzusetzen, in das ein Bauwerksinformationsmodell überführt werden kann. Sie bedienen sich dabei dem Prinzip der Objektabs-

traktion im Rahmen einer objektrelationalen Abbildung (Vgl. Kapitel 2.4.3). Die Struktur kann grob aufgeteilt werden in einen Teil, der das IFC-Schema dokumentiert und einen Teil, der die Datensätze des Bauwerks verwaltet (Vgl. Tabelle 2-7).

Tabelle 2-7: Auszug Datenbankschema IfcSQL
Quelle: Bock und Eder (2022); eigene Darstellung

Namensraum	Tabellenname	Bedeutung
ifcSchema	.Type	Haupt-Schema-Tabelle
	.EntityAttribute	Attributdefinitionen je Entität
	.EnumItem	Enumerationstypen
	.SelectItem	Selektionstypen
ifcInstance	.Entity	Haupt-Instanzen-Tabelle
	.EntityAttribute*	Reihe an Tabellen für Attributwerte
ifcProject	.Project	Projektinformationen
ifcUnit	.Unit	Einheiten

Alle Instanzen, egal welcher Entität, werden grundsätzlich in eine Tabelle geschrieben. Diese besteht aus einer Spalte für eine eindeutige Kennung als Primärschlüssel und einer Fremdschlüsselspalte für die Entitätsdefinition. Letzterer verweist auf eine Tabelle mit Entitätsdefinitionen nach gültigem IFC-Schema. Für das Management von Attributen bestehen separate Tabellen, für jede unterschiedliche Art von Datentyp im IFC-Schema eine eigene. Nach Analyse von Bock und Eder (2022) zählen dazu aktuell 18 Datentypen. Jedes Attribut der Intanzen erhält einen Eintrag in der entsprechenden Tabelle. Der Eintrag besteht aus jeweils einer Spalte für den Fremdschlüssel, der angibt, auf welche Instanz in der Instanzentabelle sich das Attribut bezieht, die Position des Attributs in der Entitätsdefinition und dem tatsächlichen Attributwert.

Aktuell liegt keine dokumentierte Anwendung des IfcSQL-Konzeptes vor. Grund dafür könnte die sehr abstrakte Herangehensweise bei der Tabellenstrukturierung sein. Zwar ist im Gegensatz zu BIM Server das Abfragen von Daten mit SQL möglich, allerdings sorgt die Struktur für mitunter umständliche Query-Inhalte. Um beispielsweise alle Wände zu filtern und deren Attribute in lesbarer Form auszugeben, sind Suchanfragen auf eine Vielzahl von Tabellen nötig (siehe Abbildung 2-10).

Finde die ID für die Entität IfcWall	*ifcSchema.Type*
Finde alle Instanzen mit jener ID	*ifcInstance.Entity*
Über jede der Instanzen	
Über jede Attributtabelle	*ifcInstance.EntityAttribute**
Finde alle Attribute mit der ID der Instanz	*ifcInstance.EntityAttribute**
Über jedes Attribut	*ifcInstance.EntityAttribute**
Finde die Attributdefinition anhand der Entität ID und der Attributposition	*ifcSchema.Type & ifcSchema.EntityAttribute*
Ausgabe Attributname und -wert	*Konsole*

Abbildung 2-10: Pseudocode Struktogramm Wand-Query in IfcSQL

Bock und Eder (2022) entscheiden sich bewusst gegen eine Aufspaltung der Instanzentabelle auf mehrere, entitätsabhängige Tabellen mit dem Verweis auf die über 750 existierenden Entitäten im IFC-Schema. Eine Tabelle für jede Entität führe zu Verwirrung und schlechter Handhabung. Die Entwickler haben mit IfcSQL gezeigt, dass die Daten eines Bauwerksinformationsmodells in eine relationale Datenbank eingepflegt werden können. Inwiefern das Datenbankschema den Ansprüchen des IFC-Metamodells gerecht wird, ist mindestens diskutabel. In jedem Fall zeigt der Ansatz wie auch BIM Server Schwächen in puncto Datenzugänglichkeit und -management auf.

2.5.3. OBJEKT-RELATIONALER IFC-MODELL SPEICHER

Dass Bock und Eder (2022) sich für eine relationale Datenbank als Grundgerüst entschieden haben, ist nicht ohne Grund. Seit über drei Jahrzenten bilden sie „die dominierende Form der strukturierten Datenhaltung in Anwendungssystemen“ (Saake, 2018). Nicht automatisch ist aber daher die relationale Datenbank auch die optimale Lösung für eine neue Anwendung. Das IFC-Schema ist stark geprägt durch einen objektorientierten Ansatz. Ein Datenhaltungsprinzip, dass diesen Ansatz unterstützt, verspricht Vorteile. Die rein objektorientierten DBMS konnten seit ihrer Einführung im Vergleich zu relationalen DBMS nicht überzeugen. Das relationale DBMS unterstützt jedoch in seiner reinen Form nicht den objektorientierten IFC-Ansatz. Forschende der Wuhan Universität entwickelten daher eine Modelltransformation für eine objekt-relationale Datenbank, die Gebrauch beider Ideen macht (Li et al.). Als Datenbankanbieter fungiert Oracle. Das Grundgerüst wird gelegt durch ein Mapping der primitiven EXPRESS-Datentypen auf Oracle interne Datentypen wie zum Beispiel STRING auf VARCHAR2. Enumerationen werden vereinfacht ebenfalls als normale VARCHAR2 angegeben, ohne die Restriktion der Werteliste zu übernehmen. Die Deklarierung der EXPRESS-Entitäten erfolgt durch die Einführung von strukturierten Typen, dem objektrelationalen Pendant der Klasse.

Ihnen werden Attribute und deren Datentypen entsprechend der EXPRESS-Definition zugeteilt.

Quelltext 2-1: Strukturierter Typ IfcRoot nach Li et al.
Quelle: Li et al. , S. 626; eigene Darstellung

```
CREATE OR REPLACE TYPE IfcRoot AS OBJECT (
     GlobalId     VARCHAR2(100),
     OwnerHistory IfcOwnerHistory,
     Name         IfcLabel,
     Description  VARCHAR2(40000)
);
```

Für die strukturierten Typen und auch die anderen Datenbankobjekte geben Li et al. keine Dokumentation an. Lediglich Ausschnitte werden vorgestellt, beispielsweise für den strukturierten Typen IfcRoot (siehe Quelltext 2-1). Dieser ist aber weder konsistent mit dem vorgeschlagenen Datenbankschema, noch wirklich mit dem IFC-Metamodell. So wird die GlobalId als Folge von Buchstaben der Größe 100 angegeben wird, obwohl es sich nach EXPRESS-Definition um einen String mit fester Größe von 22 Zeichen handelt (buildingSMART, 2020a). Dem Attribut Name ist ein definierter Typ IfcLabel zugewiesen, obgleich dieser nach vorgeschlagener Abbildung, analog zu Description, ein VARCHAR2(255) sein soll. Über die im IFC-Metamodell definierten Entitäten hinaus, entwickeln Li et al. zudem einen Typen mit dem Namen Reference. Dieser sei zur Referenzierung bestehender Objekte gedacht und hat die Attribute TableName und ID, mit denen jedes Objekt identifiziert werden kann. Damit wiedersprechen sie ihrem eigenen Anspruch, die Beziehung zwischen zwei Objekten durch eine Attribuierung des einen mit dem Typen des anderen umzusetzen. Welcher Ansatz letztlich verfolgt wurde, ist nicht klar.

In dem objektrelationalen Ansatz können Objekte erzeugt werden, sollen aber an eine Relation gebunden werden. Es gibt die Möglichkeit von Objekttabellen, die gänzlich der Speicherung von Objekten dienen, und Objektspalten, bei denen Objekten nur in Form einer Spalte an eine Relation gebunden wird. Li et al. entscheiden sich für den zweiten Ansatz, wodurch die Möglichkeit einer einfacheren Tabellenstruktur folgt. Für alle von IfcRoot abgeleiteten Objekte existieren genau drei Tabellen, den drei direkten Subtypen entsprechend. Jede Tabelle trägt dabei die selben Attribute: OID, Instance und Entity-Name. Objektreferenzen werden in das Attribut Instance gespeichert. Für die erste Tabelle mit Namen Object ist der Datentyp IfcObjectDefinition, für die Tablle Property IfcPropertyDefinition und die letzte Tabelle namens Relation trägt IfcRelationship. Inbegriffen sind jeweils auch alle Subtypen. Die Objekte der Ressourcenebene des IFC-Schemas werden in die Tabelle Resource geschrieben. Dafür ist ein zusätzlicher strukturierter Supertyp notwendig, hier IfcResource genannt, von dem alle anderen Ressourcentypen erben. Dieser Typ wird anschließend als Datentyp für die Instance-Spalte der

Tabelle Resource genutzt. Aufgrund des Übergewichts der Daten für die Ressourcenebene, werden noch zwei weitere Tabellen, einmal für Instanzen vom Typ IfcRepresentationItem und die andere für kartesische Punkte generiert (siehe Tabelle 2-8).

Tabelle 2-8: Objektrelationale Tabellenstruktur nach Li et al.

Name	Inhalt
Object	Instanzen von Subtypen von IfcObjectDefinition
Relationship	Instanzen von Subtypen von IfcRelationship
Property	Instanzen von Subtypen von IfcPropertyDefinition
Representation	Instanzen von Subtypen von IfcRepresentationItem
Point	Punkte im dreidimensionalen Raum
Resource	Instanzen verbliebener Typen

Den vorgestellten Ansatz haben Li et al. anhand dreier Beispielgebäude aus der Autodesk Revit 2023 Applikation validiert. Dabei kommen sie allerdings auf, nach eigener Aussage, wenig zufriedenstellende Speicher- und Ladezeiten (siehe Tabelle 2-9).

Tabelle 2-9: Speicher- und Ladezeiten nach Li et al. (2016)
Quelle: Li et al. , S.629; eigene Darstellung

Elemente	Speicherzeit [min]	Ladezeit [min]
425	3	2
102285	166	143
660952	669	602

Grundsätzlich wurde in dem Projekt der Wuhan Universität sehr sinnvoll Gebrauch gemacht von den Vorteilen der objektorientierten Erweiterungen relationaler DBMS. Es muss allerdings davon ausgegangen werden, dass leider nicht die Beziehungsmodellierung konform zum EXPRESS-Format vollzogen wurde. Davon abgesehen existiert mit den strukturierten Typen aber eine sehr IFC-nahe Datenstruktur. Inwiefern der Ansatz von Objektspalten die technologisch beste Option ist, muss weiter erforscht werden.

2.5.4. IFC WEBSERVER

Unter dem Namen IFC WebServer läuft seit 2010 die Entwicklung eines Modellservers für Bauwerksinformationsmodelle nach IFC-Standard (Ismail, 2023). Teil des Projektes ist eine Neo4j Graphendatenbank zur Verwaltung von Modelldaten. Informationen eines IFC-SPF werden zunächst in CSV-Dateien umgewandelt und anschließend eine Reihe, gegen die Datenbank laufende, Befehle formuliert zur Erstellung von Knoten und Kanten (siehe Abbildung 2-11). Vorgeschaltet ist eine Anwendungsoberfläche zur Auswahl zu übernehmender Entitäten.

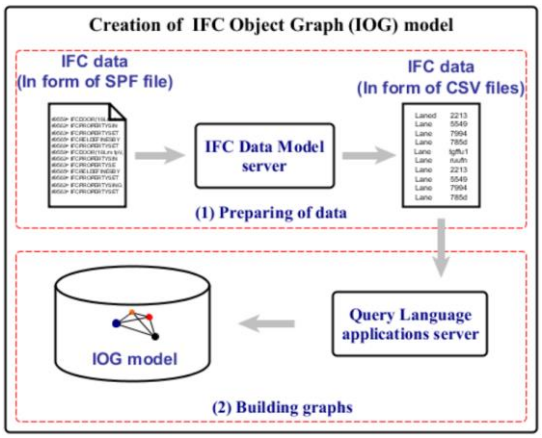


Abbildung 2-11: Arbeitsablauf IFC WebServer Graphendatenbank
Quelle: Ismail et al. (2017), S. 8

Die Graphendatenbank verfolgt den LPG-Ansatz. Informationen können also in Tripelform und als Knoten- und Kanteneigenschaften gespeichert werden. Darüber hinaus besteht in der Funktion des Labels die Möglichkeit geeignete Gruppierungen von Knoten vorzunehmen. Bei der Modelltransformation treffen Ismail et al. (2017) einige Vereinfachungen. So wird ein Großteil von objektifizierten Beziehungen aufgelöst zu einfachen Beziehungen in Form von Kanten. Die Subtypen von IfcRelDefines und die darüber verbundenen Eigenschaften werden alle als Kante isDefinedByProperties modelliert. Auf der anderen Seite sind zusätzliche Knoten eingeführt, um spätere Abfragen zu vereinfachen. Alle tragenden Wände erhalten beispielsweise eine Verbindung zu einem neuen Knoten „Load bearing walls“. Die getroffenen Vorkehrungen vereinfachen mitunter zwar die Handhabung der verwalteten Daten, stellen aber eine Abwendung vom IFC-Metamodell dar. Dadurch ist die langfristige Verwaltung ohne detaillierte Dokumentation zumindest fragwürdig. Nichtsdestotrotz zeigen Ismail et al. (2017), dass Graphen sich grundsätzlich eignen für die Verwaltung von IFC-basierten Modelldaten.

Die vorgestellten Ansätze zum datenbankbasierten Informationsaustausch haben angedeutet, welches Potential in der Modelltransformation steckt. Den Weg in die Praxis haben die dateilosen Methoden allerdings noch nicht geschafft. Die Gründe dafür sind vielseitig: nicht standardisierte Abfragemöglichkeiten, komplizierte Datenstrukturen, fehlender Bezug zum IFC-Metamodell und schlechte Performanz, um einige zu nennen. Es bedarf daher weiterer Forschung bereits erprobter Ansätze und Erforschung von neuen Ansätzen für andere Datenbankmodelle.

3 METHODIK

Hauptziel der Arbeit ist die Untersuchung von Datenbankkonzepten und deren Anwendungsmöglichkeiten für die dateilose Verwaltung von IFC-konformen Bauwerksinformationsmodellen. Inbegriffen ist der Versuch, das allgemeine EXPRESS-Datenmodell auf mögliche Datenbankmodelle abzubilden. Darauf aufbauend ist anschließend zu untersuchen, inwiefern ein konkretes BIM-Modell über ein DBMS angelegt, verwaltet und spezifische Informationen ausgegeben werden können. Dafür sollen die Daten eines bereits bestehenden Modells, serialisiert mittels des SPFF, ausgelesen und in die Datenbank geschrieben werden. Eingeschränkt ist die Untersuchung auf das relationale Datenbankmodell, ggfs. mit objektorientierten Erweiterungen, sowie das Graphendatenbankmodell. Als RDBMS bzw. ORDBMS kommt das Produkt Oracle Database zum Einsatz. Für den graphenbasierten Ansatz wird das System Neo4j verwendet. Über die DBMS hinaus sind einige Anwendungen und Programmierschnittstellen, insbesondere für die Transformation eines IFC-SPF in eine Datenbankinstanz nötig. Die relevanten davon, sowie die DBMS werden nachfolgend vorgestellt. Zunächst erfolgt aber eine Einordnung der Datenbankmodelle im Kontext des IFC-Metamodells und eine vorangestellte Inhaltsanalyse von Bauwerksinformationsmodellen.

3.1. ANALYSE DES MODELLIERUNGSUMFANGS

In der aktuellen Version IFC4 umfasst das Metamodell 776 Entitäten, 130 definierte Typen, 207 Enumerationen und 60 Selektionstypen, in Summe 1173 Definitionen. Mit kommenden Erweiterungen, insbesondere auf dem Infrastrukturbereich, steigt die Zahl weiter an. Die Version IFC4.3, aktuell im ISO-Standardisierungsprozess, umfasst bereits 1316, also fast 150 Definitionen mehr als noch IFC4 (buildingSMART, 2023). Kritiker führen die Fülle oftmals als großen Kritikpunkt an dem IFC-Konzept an. Tatsächlich wird aktuell allerdings nur ein Bruchteil davon auch in der Praxis verwendet. Haupterzeugungspunkt eines Bauwerkinformationsmodells ist nach wie vor eine Modellierungssoftware. Die buildingSMART Gruppe zertifiziert Softwarehersteller, wenn sie nachweisen können, dass ein bestimmter Modellierungsumfang sowohl bei Export als auch Import unterstützt wird (buildingSMART, 2022b). Für IFC4 ist der Umfang zusammengefasst als Model View Definition mit dem Namen Reference View (RV) in der Version 1.2 (buildingSMART, 2020b).

Tabelle 3-1: Datentypen IFC4 Reference View 1.2

Datentyp	IFC4	RV1.2
Entität	776	412
davon abstrakt	123	74
Definierter Typ	130	70
Enumeration	207	131
Selektion	60	20

Von 1173 ursprünglichen Definitionen verbleiben 663 (siehe Tabelle 3-1). Es sei an der Stelle aber darauf hingewiesen, dass es sich um eine Vorgabe mindestens zu unterstützender Definitionen handelt. Eine freiwillige Erweiterung von Seiten der Hersteller ist möglich und wünschenswert. Die größte absolute Verringerung findet bei den Entitäten statt. Ein Großteil der Streichungen der anderen Datentypen steht auch unmittelbar mit den fehlenden Entitäten im Zusammenhang. Beispielsweise entfallen solche Selektionstypen, die eine Auswahl an nun nicht mehr vorhandenen Entitäten beschreiben. Datentypen speziell für ein Attribut einer weggelassenen Entität sind ebenso obsolet. Auf der Ressourcenebene finden die größten Entitäts-Streichungen statt (siehe Tabelle 3-2).

Tabelle 3-2: Anzahl Entitäten des Reference View je Ebene

Ebene	IFC4	RV1.2
Fachbereichsebene	198	151
Interoperabilitätsebene	100	78
Kernebene	121	68
Ressourcenebene	357	115

Zusammengefasst lässt sich erahnen, dass von den möglichen Definitionen in IFC eigentlich nur ein geringer Teil für ein konkretes Bauprojekt genutzt wird oder werden kann. Im Folgenden wird daher noch ein spezifisches Informationsmodell untersucht. Es handelt sich um das Modell des zweiten Obergeschosses eines Bürogebäudes am Nürnberger Ei in der Dresdener Südvorstadt, deren Räumlichkeiten vom Institut für Bauinformatik genutzt werden und das nachfolgend CIB-Modell genannt wird. Enthalten sind Bauteile und Informationen zum Rohbau inklusive Fenster und Türen, Elemente der Inneneinrichtung, sowie eine Reihe von Sensoren und Aktuatoren zur Gebäudeautomation (Vgl. Abbildung 3-1). Außerdem sind Bestandteile der TGA modelliert. Dazu zählen Heizstrahler, ein Heizleitungssystem und Sanitäreinrichtungen. Frisch- sowie Abwasserleitungssysteme sind ebenso wenig enthalten wie der Bereich Elektro.

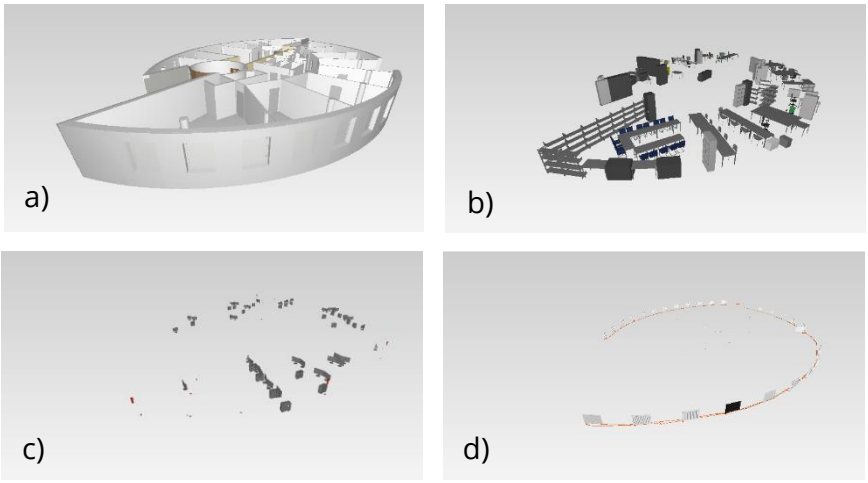


Abbildung 3-1: Teilmodelle CIB-Modell mit a) Architektur, b) Inneneinrichtung, c) Gebäudeautomation und d) TGA

Als Modellierungswerkzeug kam Revit 2023 zum Einsatz (Autodesk Inc., 2023). Die Autodesk Anwendung besitzt eine voreingestellte Abbildung von Revit-Typen, auch Kategorien genannt, auf IFC-Entitäten. Für einige Typen lautet die Zuordnung IfcBuildingElementProxy, beispielsweise die Revit-Kategorie Elektrische Ausstattung. Es handelt sich bei IfcBuildingElementProxy um eine ungenügende Abhilfe und Hilfskonstruktion, die nur für nicht in IFC enthaltene Definitionen genutzt werden soll. Andere Typen werden sogar von Werkseinstellung her gar nicht zugeordnet, darunter sämtliche Elemente der Strukturanalyse, beispielsweise der Typ Wand-Analytisch.

Revit erlaubt daher dem Nutzer die Abbildung auf drei Wegen anzupassen (Autodesk Inc., 2022). Entweder wird die Zuordnung für den gesamten Typen, beispielsweise alle Wände, oder exemplar- bzw. familienweise, für bestimmte Wände, geändert. Ersteres kann in den allgemeinen IFC-Exporteinstellungen vorgenommen werden. Der zweite Ansatz wird über eine zusätzliche Elementeigenschaft realisiert. Für die gewünschten Typen ist die Eigenschaft IfcExportAs anzulegen und mit dem Namen einer existierenden IFC-Entität zu befüllen. Dies kann für eine gesamte Familie gleich oder bei jeder Instanz unterschiedlich erfolgen. Bei anschließendem Export werden die Elemente entsprechend der Angabe abgebildet. Die Werkseinstellungen für den IFC-Export entsprechen den Vorgaben von buildingSMART und der Reference View. Die beiden vorgestellten Optionen zur Änderung erlauben aber theoretisch eine sehr detaillierte Abbildung. Problematisch ist dabei, dass die in Revit erzeugten Eigenschaften und Informationen nicht zwangsläufig zur IFC-Entität passen. Es ist daher große Vorsicht geboten und tiefgehendes Wissen über das Modellierungsschema nötig, wenn der elementare Level der Grundeinstellungen erweitert wird. Darüber hinaus fällt auf, dass Elemente einiger Kategorien trotz Ausnutzen der Mapping-Funktionen nicht nur nicht IFC-konform, sondern gar nicht exportiert werden können. Dazu zählen unter anderem Elemente der stati-

schen Analyse, z.B. Lagerbedingungen, aber auch zweidimensionale Architekturelemente wie Modelllinien.

Für die vier Teilmodelle des CIB-Modells ist ein Kompromiss aus Komplexität und Detaillierungsgrad bei den Exporteinstellungen getroffen. Eine Analyse der exportierten Entitäten bestätigt den Verdacht, dass nur ein Bruchteil des IFC-Schemas tatsächlich verwendet wird (Vgl. Tabelle 3-3). Die Anzahl von Instanzen ist dabei offensichtlich nicht abhängig von der Anzahl unterschiedlicher Entitäten.

Tabelle 3-3: Verschiedene Entitäten und Instanzen in CIB-Modell nach Teilmodell

	Entitäten	Instanzen
Architektur	91	25420
Inneneinrichtung	65	71315
Gebäudeautomation	61	3754
TGA	108	596550
Gesamt	132	697039

Im Rahmen der Arbeit wird nur das Teilmodell Architektur mit dem Rohbau inklusive nichttragender Wände, Fenstern und Türen untersucht. Da die verfolgten Transformationsansätze allgemein gehalten sind, ist eine Erweiterung auf die anderen Aspekte möglich. Das Gros an Instanzen fällt bei der Beschreibung von Geometrie und (Re-)Präsentation auf der Ressourcenebene an (Vgl. Tabelle 3-4). Allein die Entität IfcIndexedPolygonalFace weist 15164 Instanzen auf und macht damit einen Anteil von fast 60% der insgesamt 25469 Instanzen aus. Es handelt sich um die Beschreibung einzelner planarer Teilflächen über die Angabe indizierter Koordinatenpunkte, die zusammen einen Körper ergeben.

Tabelle 3-4: Entitäten und Instanzen im Teilmodell Architektur

	Entitäten	Instanzen	Anteil an Gesamtinstanzen
Fachbereichsebene	3	14	< 1%
Interoperabilitätsebene	19	213	1%
Kernebene	16	2963	12%
Ressourcenebene	53	22230	87%

Ergebnis der Analyse ist ein Überblick darüber, welcher Modellierungsumfang tatsächlich abgebildet werden muss und in welcher Größenordnung die zu verwaltenden Datensätze zu erwarten sind.

3.2. OBJEKTRELATIONALER ANSATZ

Das Ziel ist es, die Daten des beschriebenen Bauwerkinformationsmodells in eine objektrelationale Datenbank von Oracle zu überführen. Dafür muss zuvor eine Schemastruktur entwickelt werden, die in der Lage ist, solche Daten sinnvoll zu speichern. Das Hauptkriterium soll die Treue zu den EXPRESS-Definitionen des IFC-Metamodells sein. Im ersten Schritt wird das rein relationale Datenbankmodell untersucht. Es ist etabliert und hat sich in der Vergangenheit als effizient, robust und verlässlich bewiesen. Da das IFC-Schema objektorientiert ist, bedarf es in der Theorie nur einer objektrelationalen Abbildung. Das Konzept der Objektabstraktion wie in IfcSQL (Vgl. Kapitel 2.5.2) wird aufgrund des komplizierten Speichersystems und der nur schwer erkennbaren IFC-Struktur nicht verfolgt.

3.2.1. OBJEKTRELATIONALE ABBILDUNG DES IFC-METAMODELLS

Ein erster Ansatz der objektrelationalen Abbildung beschreibt eine Relation je Vererbungsbaum des Objektmodells. Was allgemein möglich ist, wird im vorliegenden Fall unpraktikabel. Das Herzstück des IFC-Metamodells ist der IfcRoot-Vererbungsbaum. Alle Entitäten der Kernebene, Interoperabilitätsebene und Fachbereichsebene stammen von ein und der selben Entität ab, IfcRoot. Dazu zählen sämtliche allgemeinen Objektdefinitionen, Merkmaldefinitionen und Beziehungen. Ansatz Eins sieht vor, alle Instanzen dieser Entitäten in ein und die selbe Relation zu überführen. Dafür muss die Relation sämtliche Attribute, die im Verlaufe des Vererbungsbaums auftreten, abbilden können. Es ist vorstellbar, dass unter diesen Voraussetzungen eine Relation mit einer schier unverwaltbaren Menge an Attributen entsteht, von denen ein Tupel jeweils nur einen Bruchteil nutzt. Für das verhältnismäßig kleine Teilmodell Architektur des CIB-Modells liegen 38 konkrete Subtypen von IfcRoot vor (Vgl. Tabelle 3-4). Eine Analyse der unterschiedlichen Subtypen ergibt 70 abzubildende Attribute. Abgesehen von der nicht erstrebenswerten horizontalen Skalierung, ist auch das Auslesen der einzelnen Datensätze unpraktikabel. Ohne zusätzliche Relationen, die das IFC-Metamodell dokumentieren, ist es nicht möglich die tatsächlichen Attribute des abgebildeten Objektes zu identifizieren. Jedes Tupel weist automatisch eine Vielzahl von NULL-Werten auf. Welche davon NULL sind, weil sie für die zugrundeliegende Entität nicht existieren und welche NULL sind, weil sie optional sind und nicht ausgefüllt wurden, ist unklar. Sollen zudem in Zukunft alle Entitäten des IFC-Metamodells abbildbar sein, erhöhte sich die Zahl an unterschiedlichen Attributen auf 237. Diese Herangehensweise ist nicht empfehlenswert.

Der zweite Ansatz bildet eine Relation pro Klasse. Dadurch hat jede Relation exakt die Attribute, die für die entsprechende Entität auch im IFC-Metamodell definiert sind. Die Vielzahl von Attributen des Vererbungsbaums werden aufgelöst auf die entsprechenden Ebenen. Es existieren jedoch einige Entitäten in IFC, die aktuell keine Attribute zum Vererbungsbaum hinzufügen, beispielsweise IfcBuildingElement. Diese auszulassen ergibt

aus Effizienzaspekten Sinn, führt jedoch zu Inkonsistenz mit dem Metamodell. Allgemein ist dieser Ansatz vor allem zielführend bei kleineren Vererbungsstrukturen. Am Beispiel der Entität IfcWall wird das Prinzip ad absurdum geführt. Es sind sich referenzierende Tupel in den Relationen IfcRoot, IfcObjectDefinition, IfcObject, IfcProduct, IfcElement, IfcBuildingElement und schlussendlich IfcWall nötig, um ein einzelnes Objekt abbilden zu können. Der große Vorteil ist dafür aber, dass auf schnellem Wege alle Instanzen einer Entität mitsamt seinen Subtypen abfragbar sind. Das Datenbankschema braucht nur auf die Primärschlüssel der in der entsprechenden Relation enthaltenen Tupel abgesucht werden. Nachteilig ist jedoch, dass für das Auslesen einer konkreten Instanz komplexe Join-Befehle und nötig sind.

Ein letzter Ansatz sieht vor, nur die instanziierbaren Klassen auf Relationen abzubilden. Die Attribute abstrakter Supertypen werden in den konkreten Subtypen übernommen. Das Mehrfachanlegen von Tupeln wird dadurch zwar unterbunden, allerdings sind dafür nun die Attribute der abstrakten Entitäten redundant gespeichert. Der große Vorteil dieses Ansatzes ist die direkte Interpretierbarkeit der Tupel als IFC-konforme Objekte. Mit dem Ziel, das EXPRESS-Datenmodell so gut wie möglich zu serialisieren, wird im Rahmen der Arbeit der dritte Ansatz weiterverfolgt.

3.2.2. MODELLIERUNG DER BEZIEHUNGEN

In EXPRESS erfolgt eine Beziehung zwischen Klassen, indem die eine Klasse ein Attribut aufweist, dessen Datentyp die andere Klasse bildet. Relationen hingegen werden durch Primärschlüssel-Fremdschlüssel-Paare verbunden. Anhand der Beziehung zur räumlichen Zuordnung von Elementen, IfcRelContainedInSpatialStructure, wird nachfolgend das Vorgehen zur Abbildung der Beziehungsmodellierung erklärt (Vgl. Abbildung 3-2). Die objektifizierte Beziehung ordnet genau einer räumlichen Struktur, beispielsweise einem Bauwerksgeschoss, eine Menge an Elementen, Wände, Stützen und Ähnliches, zu. Konkret verbindet es eine Instanz eines Subtypens von IfcSpatialElement mit einem Set S[1:?] an Instanzen von Subtypen von IfcProduct. Die entsprechenden Attribute von IfcRelContainedInSpatialStructure lauten RelatedElements für das Set und RelatingStructure für die Raumstruktur. Die EXPRESS-Notation ist bekannt für explizit inverse Beziehungen. Nicht nur die objektifizierte Beziehung weist entsprechende Attribute zu anderen Klassen auf, sondern auch die Klassen andersherum. Dabei gibt es auf Seiten der enthaltenen Elemente eine Abweichung. Nicht IfcProduct, sondern IfcElement, der direkte Subtyp, weist das inverse Attribut ContainedInStructure als Set S[0:1] auf. Die Kardinalität zeigt, dass jede Instanz eines instanziierbaren Subtypens von IfcElement in keiner oder genau einer räumlichen Beziehung stehen darf. Auf der anderen Seite kann mit dem Attribut ContainsElements und einer Kardinalität von S[0:?] ein IfcSpatialStructure in keiner oder beliebig vielen solcher Beziehungen sein.

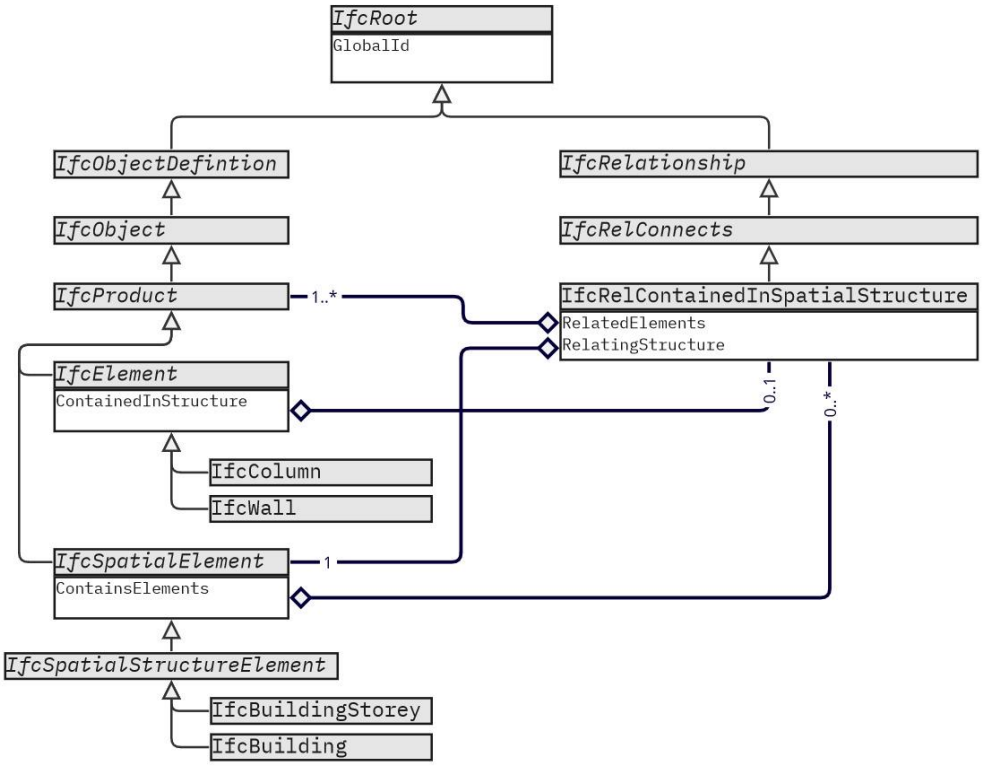


Abbildung 3-2: Auszug UML Klassendiagramm Spatial Structure Konzept

Das Datenbankschema soll nur die konkreten Typen als Relationen abbilden. Die Beziehungen im IFC-Metamodell finden aber häufig zwischen abstrakten Typen statt. Sie müssen also aufgelöst werden auf mehrere, die konkreten Klassen betreffende, Beziehungen. Es soll nicht nur die vorwärts gerichtete Navigation, sondern über inverse Beziehungen auch die rückwärtsgerichtete möglich sein. Die beschriebenen Anforderungen führen mit dem aktuellen Ansatz zur objektrelationalen Abbildung unweigerlich zu einem Bruch mit allgemeinen Datenbankdesignregeln (siehe Abbildung 3-3).

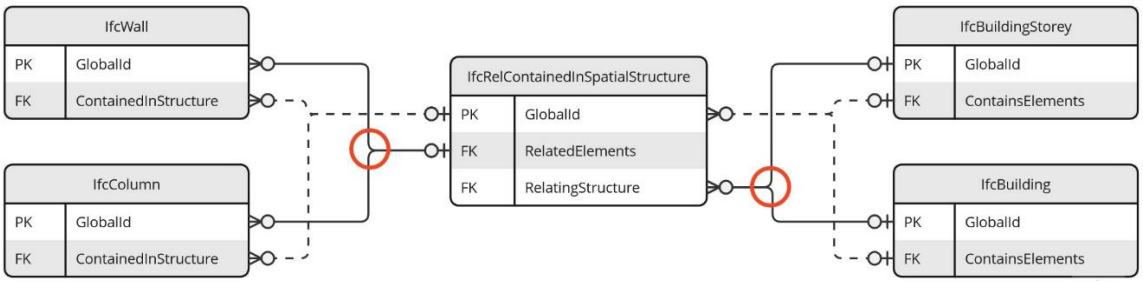


Abbildung 3-3: Auszug ERM Spatial Structure Konzept – Option 1

Die 1:n Beziehung zwischen IfcRelContainedInSpatialStructure und IfcProduct spaltet sich auf in eine Reihe von 1:n Beziehungen mit den instanziierbaren Subtypen von IfcProduct. Theoretisch sollen alle Beziehungen in ein und dem selben Attribut RelatedElements abgebildet werden. Ein Fremdschlüsselattribut kann allerdings immer nur genau eine andere Relation referenzieren. Folglich sind in der Modellierung weitere Attribute nötig, die speziell jeweils eine der möglichen Relationen adressiert (siehe Abbildung 3-4). Neben dem Bruch mit dem IFC-Schema resultiert daraus eine unübersichtliche Vielzahl von Attributen. Allein IfcProduct hat 155 nicht abstrakte Subtypen, die eine eigene Fremdschlüsselspalte fordern. Auf der anderen Seite der Relation tritt das selbe Problem auf. Dadurch, dass die allgemeine Beziehung auf konkrete Beziehungen umgeschrieben wird, muss für alle 6 nicht abstrakten Subtypen von IfcSpatialStructure ein eigenständiges Attribut definiert werden. Zusätzlich bereitet das objektorientierte Metamodell Probleme bei der Normalisierung. Eine räumliche Zuordnung, IfcRelContainedInSpatialStructure, kann beliebig viele Produkte über RelatedElements adressieren. Ein Tupel, dass eine konkrete objektifizierte Beziehung in der entsprechenden Relation abbildet, darf nach erster Normalform aber nicht mehrere Werte, hier Fremdschlüssel, für ein Attribut aufweisen. Stattdessen muss der Datensatz auf mehrere Tupel verteilt werden. Die GlobalId der Objektbeziehung kann aber nicht gleichzeitig der Primärschlüssel mehrerer Tupel sein. Es wird klar, dass eine einfache objektrelationale Abbildung auf diesem Weg nicht durchführbar ist. Abhilfe ist nur unter Inkaufnahme neuer Komplikationen möglich.

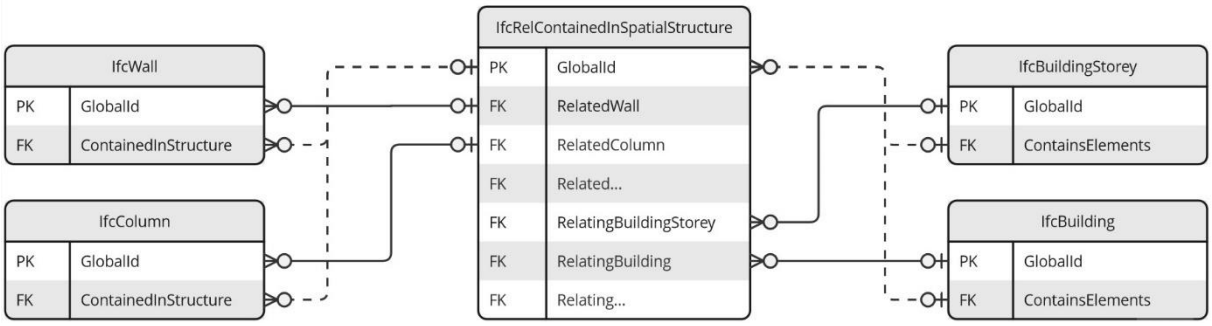


Abbildung 3-4: Auszug ERM Spatial Structure Konzept – Option 2

Die Angabe mehrerer Relationen in einem Attribut kann durch einen anderen Ansatz der objektrelationalen Abbildung umgangen werden. Durch die Abbildung aller und nicht nur der konkreten Klassen sind Beziehungen auf einer allgemeineren Ebene möglich. Über den gesamten Vererbungsbaum existieren Tupel mit der selben GlobalId als Primärschlüssel. Sie referenzieren sich gegenseitig und auf jeder Ebene können spezifische neue Attribute hinzufügefügt werden. Im Fall von IfcRelContainedInSpatialStructure wird dann nur noch eine einzige Beziehung zu der Relation IfcProduct hergestellt. Die hier referenzierten Tupel sind allerdings nicht direkt als Objekt interpretierbar. Stattdes-

sen muss ggfs. ein Großteil des Datenbankschemas nach Datensätzen mit der gleichen GlobalId abgesucht werden. Das Zusammenfügen jener Tupel mit Join-Befehlen ergibt dann eine als Objekt interpretierbare Informationsmenge. Der Vorteil in der Modellierung von Beziehungen geht also einher mit einer komplexeren Datenbankstruktur. Für das Problem mengenartiger Attributwerte braucht es auch bei diesem Ansatz noch weitere Schritte. Denkbar sind Hilfsrelationen, die die Mengen abbilden. Alternativ führt auch eine leichte Abweichung vom IFC-Metamodell zum Ergebnis. Eine 1:n Beziehung zwischen Relation A und Relation B wird in der Regel durch einen Fremdschlüssel in Relation B gelöst. Dass, wie in der übertragenen EXPRESS-Definition, auch Relation A einen Fremdschlüssel erhält, führt unweigerlich zu einem mengenartigen Attribut. Folgerichtig muss dieses Attribut entfernt werden. Es entsteht eine Mischform aus IFC-Konformität und Designregeln (Vgl. Abbildung 3-5).

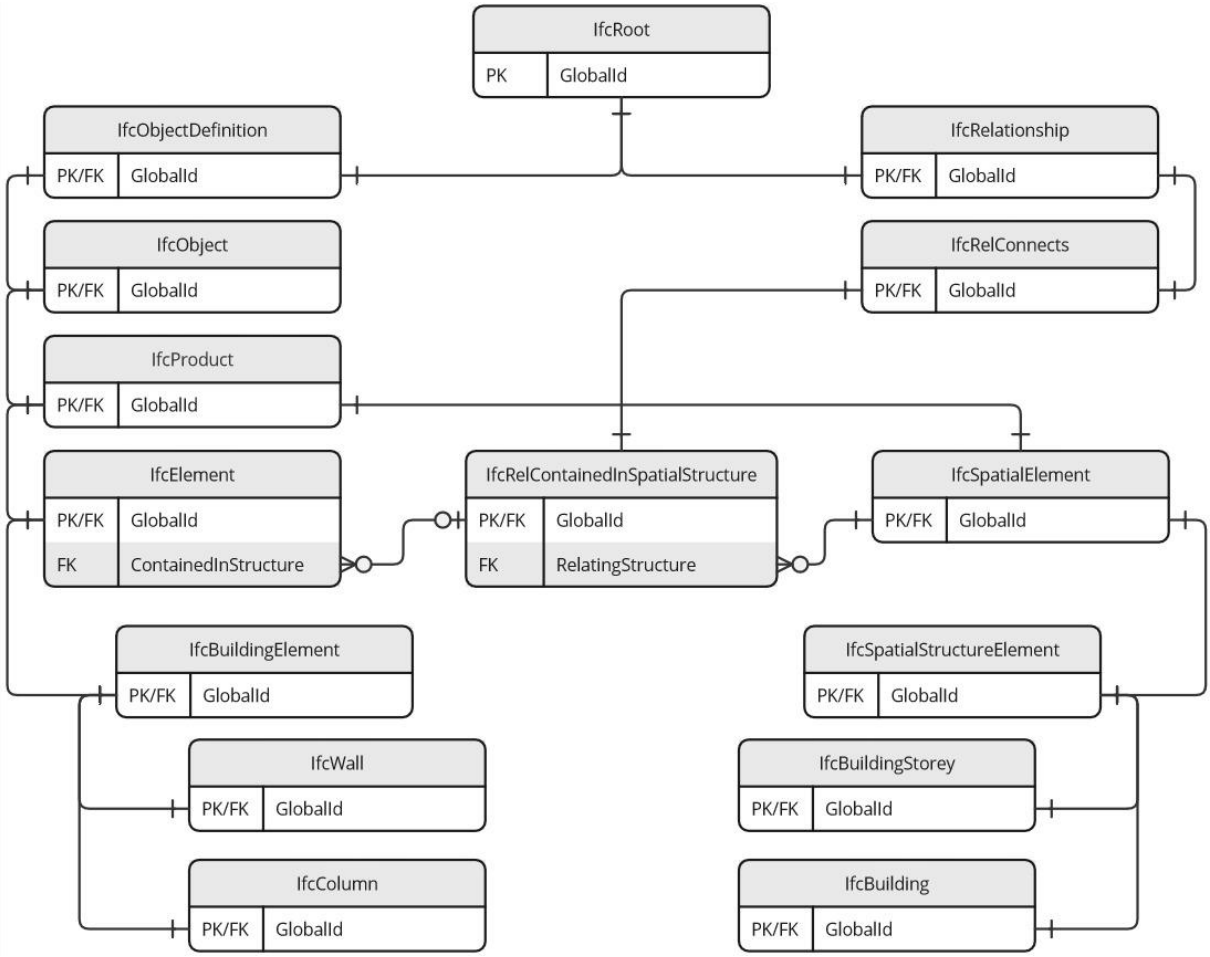


Abbildung 3-5: Auszug ERM Spatial Structure Konzept – Option 3

Die vorgestellte Lösung ist aus vielen Gesichtspunkten nur bedingt zufriedenstellend. Der eigene Anspruch an hohe IFC-Konformität ist verloren gegangen. Für das Beispiel IfcRelContainedInSpatialStructure liegt sogar ein unüberwindbarer Fehler vor. Der Fremdschlüssel für die Beziehung zu der Menge an Elementen ist mit ContainedInStructure für die abstrakte Klasse IfcElement definiert. Nach EXPRESS-Notation für die objektifizierte Beziehung sind aber auch Elemente anderer Subtypen von IfcProduct erlaubt. Eine datenbanktechnisch korrekte Modellierung schließt entweder diese Elemente aus oder schreibt das Attribut auf den Supertypen um. Das ausgewählte Beispiel stellt dabei keine Ausnahme dar, da die inkonsistente Zwei-Wege-Modellierung an mehreren Stellen vorhanden ist.

Durch den notwendigen Wechsel des Ansatzes für die objektrelationale Abbildung entstehen neue Herausforderungen. Dazu zählt die erhöhte Anzahl von Relationen und insbesondere Beziehungen, der Aufwand zur Datenextraktion und vor allem der Verlust der direkten Interpretierbarkeit eines Tupels als Objekt. Stattdessen müssen hierfür über Join-Befehle diverse Tupel zusammengefügt werden. Im Rahmen der Arbeit wird die Schlussfolgerung gezogen, dass das relationale Datenbankmodell keine optimale Eignung für eine streng IFC-konforme Modellierung aufweist. Stattdessen wird die objektrelationale Mischform aus objektorientiert und relational untersucht.

3.2.3. OBJEKTORIENTIERTE ERWEITERUNGEN

Die große objektorientierte Erweiterung des relationalen Datenbankmodells ist die Möglichkeit, Informationen in Form von Objekten zu speichern. Sie sollen dabei aber nicht ungebunden in den persistenten Speicher geschrieben werden, sondern idealerweise in den Kontext von Relationen gesetzt werden. Eine Möglichkeit ist es, einem Attribut einer Relation eine Objektklasse als Datentyp zuzuweisen. Anschließend ist jedes Tupel in der Lage, ein Objekt zu speichern. Solche Attribute werden auch Objektspalten genannt. Die Alternative sind Objektrelationen, auch Objekttabellen. Dabei wird die Aufgabe der Relation gänzlich als Speichern von Objekten eines bestimmten Typens definiert. Jedes Attribut der Klasse wird auf ein Attribut der Relation abgebildet. Kurzgesagt werden beim ersten Ansatz Objekte als Spalte, beim zweiten als Zeile gespeichert (siehe Abbildung 3-6). Im Rahmen der Arbeit wird der Ansatz von Objektrelationen verfolgt. Er verspricht hohe IFC-Konformität und hohen Wiedererkennungswert bei konsistenter Datenspeicherung. Die Daten werden als Objekte gespeichert, aber durch Relationen repräsentiert. Es ist hierdurch möglich, die Attribute der Objekte durch bekannte SQL-Befehle anzulegen, zu ändern und abzufragen bei objektorientierter Datenspeicherung.

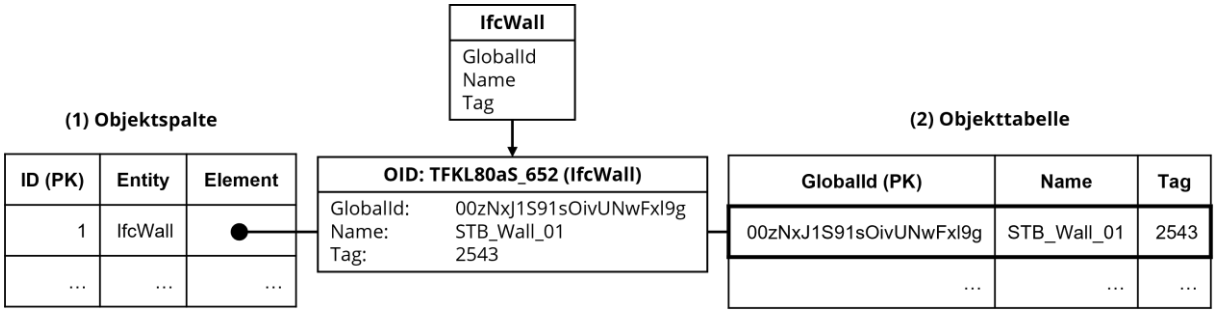


Abbildung 3-6: Konzepte Objektspalte und Objekttabelle

Der große Vorteil des objektrelationalen Ansatzes liegt in der Modellierung der Beziehungen. Ein Fremdschlüssel kann nur eine bestimmte Relation adressieren. Eine Objektreferenz hingegen bezieht sich auf Objekte einer Klasse, aber auch alle Subtypen dieser Klasse. Für das Beispiel der räumlichen Zuordnung, IfcRelContainedInSpatialStructure, kann dadurch das Problem der Vielzahl von Attributen für jeden Fremdschlüssel umgangen werden. Dafür ist es auch nicht nötig, alle Klassen in Form von Relationen abzubilden. Stattdessen werden wie angestrebt nur die konkreten als Objekttabellen realisiert. Das zweite große Problem des relationalen Ansatzes sind die mengenartigen Attribute des IFC-Metamodells. Abhilfe schafft die zweite große objektorientierte Erweiterung, die Kollektionsdatentypen. Sie sind in der Lage eine begrenzte oder unbegrenzte Menge von gleichartigen Daten abzubilden. Es kann sich um eine Menge eines primitiven Datentyps handeln, aber auch um Objektreferenzen oder erneut Kollektionen.

3.3. GRAPHENBASIERTER ANSATZ

Dem schemagebundenen relationalen Ansatz stehen eine Vielzahl von ungebundenen No-SQL-Datenbankmodellen gegenüber. Im Rahmen der Arbeit wird der graphenbasierte Ansatz untersucht. Das IFC-Metamodell ist gekennzeichnet durch stark miteinander verknüpfte Objekte. Graphen auf der anderen Seite sind insbesondere geeignet, um diese Beziehungen und Netzwerke zu modellieren, was sie zu einem idealen Ansatz zur Datenverwaltung macht. Für Graphenmodelle existieren zwei verschiedene Ansätze. Das rein Tripel-basierte RDF und der um Eigenschaften und Label ergänzte LPG (siehe Abbildung 3-7). Mit ifcOWL besteht eine von buildingSMART anerkannte Version des IFC-Schemas in der Web Ontology Language, einer ontologiebezogenen Erweiterung des RDF (Pauwels & Terkaj, 2019). Damit lassen sich IFC-Daten bereits jetzt in einer beliebigen Serialisierung des RDF formulieren. Nach dem modularen Ansatz der Web Ontology Language ist eine übergreifende Modellierung wie bei ifcOWL allerdings nicht ideal (W3C, 2023). Stattdessen werden eine Vielzahl von nicht zwangsläufig IFC-basierten Ontologien klug miteinander kombiniert, um Bauwerksinformationen darzustellen. Haupt-

ziel ist es, so wenig redundante Teilontologien zu produzieren wie möglich und eher bereits bestehende wieder zu verwenden.

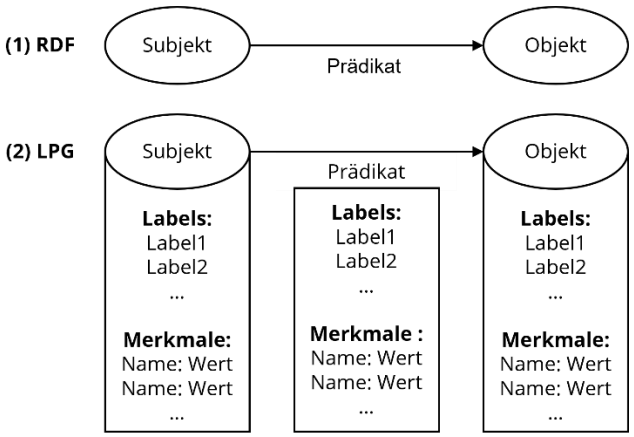


Abbildung 3-7: Graphkonzepte RDF und LPG

Im Kontext von projektbezogener Datenverwaltung hat dieser Gedanke nur begrenzt Gewicht. Vielmehr geht es um eine konsistente, effiziente und nutzerorientierte Speicherung von Informationen. Die Verwendung der Definitionen nach ifcOWL in einer RDF-orientierten Graphendatenbank ist daher vertretbar. Allerdings besteht in dem LPG-Ansatz eine Modellierung, die informationstechnisch effizienter und natürlicher ist. Attribute primitiven Datentyps werden, statt als Literal-Knoten im RDF, als Knoteneigenschaften angelegt und nur Objektreferenzen als Kanten modelliert. Dadurch wird die Menge an Komponenten drastisch reduziert. Aus einer datentechnologischen Sicht ist ein LPG daher in der Regel dem RDF überlegen (Zhu et al., 2022). Zudem besteht theoretisch die Möglichkeit, auch den Kanten Eigenschaften zuzuweisen. Ein Anwendungsfall könnte die Versionierung sein, bei der sich je nach Wert für ein Kantenattribut ein anderes Modell ergibt. Im Rahmen der Arbeit werden daher die Möglichkeiten des LPG, auch im Kontrast zum RDF untersucht.

3.4. VERWENDETE DBMS

Sowohl der objektrelationale als auch der graphenbasierte Ansatz bedürfen eines Datenbankmanagementsystems, dass den jeweiligen Ansatz unterstützt. Für das objektrelationale Modell wird das Produkt Oracle Database in der Version 12.2 verwendet (Oracle Corporation, 2016). Aufgrund der standardisierten Befehlssprache SQL für solche Systeme, ist der Ansatz aber auch auf andere Produkte anwendbar. Oracle Database kommt zum Einsatz, da die Anwendung auf dem Markt etabliert ist und verlässliche Ergebnisse erzeugt. Zudem existiert eine Vielzahl kompatibler Anwendungen und Programmierschnittstellen, die den Schemaentwurf und das Befüllen mit Daten erleichtern.

Das DBMS unterstützt sowohl relationale als auch objektrelationale Datenspeicherung. Über SQL hinaus unterstützt das DBMS die von Oracle entwickelte SQL-Erweiterung Procedural Language/Structured Query Language (PL/SQL). Sie ermöglicht Entwicklern die Erstellung von komplexen Funktionen, Prozeduren und Ausnahmehandlern innerhalb der Datenbank.

Als Management System für den LPG-Ansatz kommt Neo4j in der Enterprise Edition, Version 4.4.19, zum Einsatz (Neo4j, 2023). Es kann generische Graphen abbilden und ist Vorreiter für dieses Datenbankmodell. Da für LPGs aktuell keine standardisierte Befehlssprache existiert, ist der Austausch des Systems hier nicht trivial. Neo4j setzt seine eigene eingebaute Abfragesprache namens Cypher ein. Die deklarative Sprache befindet sich aktuell im ISO-Standardisierungsprozess und soll Grundlage sein für eine einheitliche Abfragesprache von LPGs (Grren et al., 2018). Auch deshalb wird in der Arbeit Neo4j verwendet. Die Abfragen werden in Form von Graphmustern geschrieben, wobei Knoten und Kanten als Variablen fungieren und mithilfe von Prädikaten und Bedingungen miteinander verknüpft werden können. Auf diese Weise lassen sich komplexe Abfragen erstellen, die sich auf Beziehungen und Zusammenhänge innerhalb des Graphen beziehen.

3.5. VERWENDETE PROGRAMMIERSCHNITTSTELLEN

Für die Modelltransformationen kommen verschiedene Applikationen und Programmierschnittstellen zum Einsatz. Dazu zählen Hilfsmittel für die Erstellung des objektrelationalen Datenbankschemas, sowie extern geschriebene Programmierpakete, die für den Befüllungsalgorithmus genutzt werden.

3.5.1. ORACLE SQL DEVELOPER UND DATA MODELER

Für den Entwurf des objektrelationalen Datenbankschemas kommt der Oracle SQL Developer zum Einsatz. Es handelt sich um eine Open-Source-Entwicklungsumgebung, in der SQL-Abfragen und Befehle auf eine Datenbank ausgeübt werden können (Oracle Corporation, 2022a). Zusätzlich erlaubt der Developer auch das Betrachten, Anlegen und Bearbeiten von Datenbankobjekten ohne ausgeprägte SQL-Kenntnisse über eine Software-Oberfläche. Teil des Produktes, aber auch als eigenständige Applikation erhältlich, ist der SQL Developer Data Modeler. Er bietet eine grafische Oberfläche für das Datenbankdesign (Oracle Corporation, 2022b). Hauptbestandteil des Designs ist in der Regel das logische Modell. Auf einer konzeptionellen Ebene werden Entitäten, sowie deren Attribute und Beziehungen in Form eines Entity-Relationship-Diagramms (ERD) modelliert. Davon ableitbar ist eine beliebige Anzahl an relationalen Modellen (siehe Abbildung 3-8). Die Erstellung jener umfasst die Abbildung ausgewählter Entitäten auf Tabellen, Attribute auf Spalten und Beziehungen auf Primärschlüssel-Fremdschlüssel Kombinati-

onen. Ein relationales Modell ist anschließend Grundlage für eine beliebige Anzahl an physischen Modellen. Dabei ist das physische Modell die Konkretisierung der allgemeinen Definition auf eine spezifische Datenbankstruktur wie zum Beispiel Oracle 12cR2.

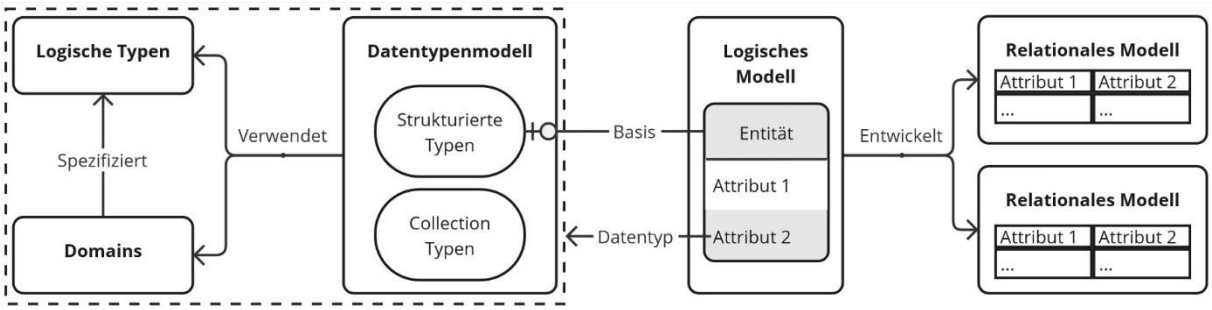


Abbildung 3-8: Auszug Architektur Oracle SQL Developer Data Modeler

Logische Typen und Domains

Kern der Modellierung sind logische Datentypen als Elementarbausteine der Datentypstruktur. Domains sind selbstdefinierte Spezifizierungen der logischen Typen. Beispielsweise wird eine generische Buchstaben-Zahlen-Kombination, VARCHAR, so auf eine bestimmte Folgengröße begrenzt und die Einstellung gespeichert. Zusätzliche Domain-Optionen sind die Angabe eines Wertebereichs, sowie einer Werteliste. Ein Wertebereich kann beispielsweise den Reellzahlentyp NUMBER auf Zahlen zwischen 0 und 10 einschränken. Die Definition einer Werteliste hat hingegen zur Folge, dass als Attributwerte ausschließlich die in der Liste eingetragenen Werte möglich sind.

Datentypenmodell

Mit Domains und logischen Typen als Grundlage wird optional ein Datentypenmodell aufgesetzt. Die Hauptbestandteile sind Kollection-Typen und strukturierte Typen. Es wird unterschieden zwischen Kollektionen mit fester Größe, ARRAY, und Kollektionen mit variabler Größe, COLLECTION. Bei beiden Typen handelt es sich um sortierte Listen mit Elementen eines gleichen Datentyps. Zu diesen Datentypen gehören logische Typen, Domains, aber auch strukturierte Typen. Letztere sind gleichzeitig die zweite große Erweiterung des relationalen Modells. Strukturierte Typen, auch Strukturtypen, sind vergleichbar mit Klassen aus dem objektorientierten Kontext. Sie unterstützen Attribuierung, sowie Vererbung und sind Baupläne zur Instanziierung von Objekten. Zusätzlich zu der relationalen Datenstruktur, in der Daten abhängig von der Relation nur als Tupel existieren, werden damit auch Objektdatenstrukturen abhängig von Typdefinitionen unterstützt. Für den objektrelationalen Ansatz bilden sie die elementare Grundlage.

Logisches Modell

Hauptentwurfsaufgabe ist das Anlegen von Entitäten im logischen Modell. Die Mindestanforderung an eine Entität sind ein Name und ein Attribut, dass als Primärschlüssel für diese Entität fungiert. Da auch auf Entitätsebene Vererbung unterstützt wird, kann dieses Attribut auf einer höher geordneten Ebene definiert sein. Als Datentyp für das Attribut muss ein logischer Typ, eine Domain oder ein Typ aus dem Datentypenmodell verwendet werden. Neben der statischen Definition einer Entität kann auch ein strukturierter Typ als Grundlage dienen. Diese Art der Entität wird auch Objekttabelle genannt und ist der angestrebte Modellierungsansatz.

Relationales Modell

Entitäten des logischen Modells werden zu Relationen im relationalen Modell. Es handelt sich um eine Konkretisierung für einen bestimmten Anwendungsfall, da häufig nicht alle im logischen Modell erstellten Entitäten für die finale Schemastruktur von Bedeutung sind. Die Modellierungsarchitektur erlaubt daher auch, mehrere relationale Modelle mit jeweils einem Anwendungszweck aus dem zentralen logischen Modell zu erstellen. Auch wenn der Übergang von Entität zu Relation der empfohlene Arbeitsablauf ist, können Relationen ohne Basis, von Grund auf modelliert werden. Davon ist in der Regel aber abzusehen aufgrund der fehlenden Replizierbarkeit.

Aus dem relationalen Modell heraus erstellt der Data Modeler auf Wunsch die benötigten Befehle, mit denen ein physisches Modell entwickelt wird. Die Befehle werden in einer Datendefinitionssprache (DDL; en: Data Definition Language) formuliert. Unterstützt werden aktuell anbieterspezifische Dialekte von SQL für DBMS-Versionen von Oracle Database, Microsoft SQL Server und IBM DB2. Die Befehle können beispielsweise über den Oracle SQL Developer auf eine leere Datenbankinstanz angewendet werden, wodurch das gewünschte Schema entsteht. Dieses kann anschließend im Data Modeler mit dem relationalen Modell gekoppelt werden. Schlussendlich enthält der Entwurf also die internen logischen Typen, Domains, ein Datentypenmodell, ein logisches Modell, beliebig viele relationale Modelle und für jedes relationale Modell beliebig viele physische Modelle.

3.5.2. PYTHON-ORACLEDB

Mit dem SQL Developer bietet Oracle eine grafische Oberfläche, um auf ein DBMS zuzugreifen. Darüber hinaus stellen sie Schnittstellen zu verschiedenen Programmiersprachen zur Verfügung. Mit ihnen ist es möglich, SQL-Befehle direkt aus der Programmierumgebung heraus auszuführen. Das erlaubt eine Vielzahl von Manipulationen, darunter die Erstellung von Datenbankobjekten wie Tabellen und Views, das Befüllen von Tabellen mit Daten und das Binden von Abfrageergebnissen an Objekte. Für die Sprache

Python läuft die Schnittstelle unter dem Namen python-oracledb, nachfolgend oracledb genannt (Oracle Corporation, 2023b).

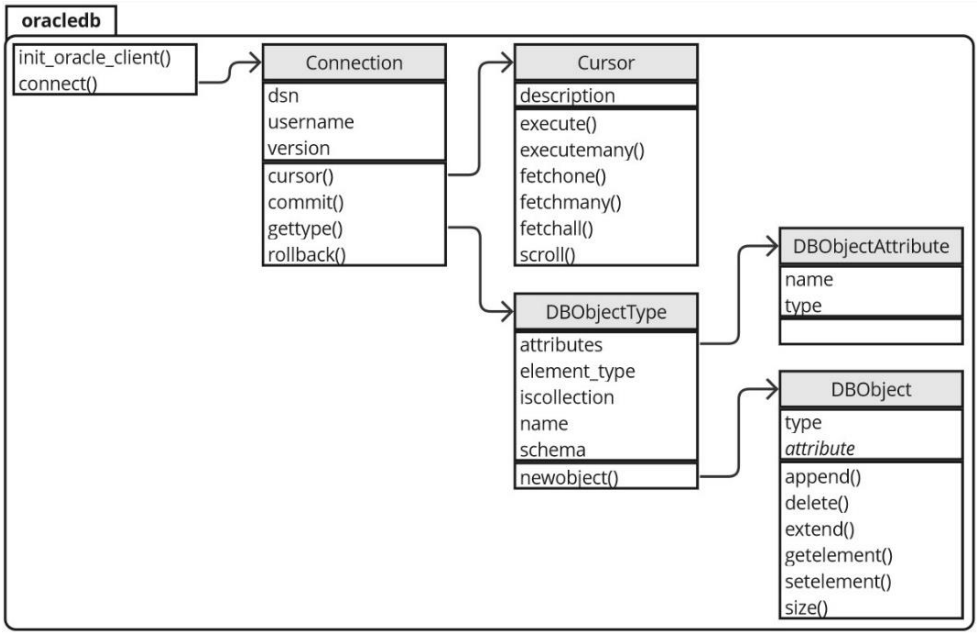


Abbildung 3-9: Auszug oracledb Modul

Grundsätzlich beginnt jede Interaktion damit, eine Verbindung zur Datenbank herzustellen. Dem vorangestellt ist die optionale Initialisierung des Oracle Clients, der erweiterte Netzwerkfuktionen bereitstellt (Oracle Corporation, 2023a). Die Methode connect() benötigt als Übergabeinformationen im Allgemeinen Informationen zu Host, Port, System-ID (SID), Benutzernamen und Kennwort. Der Host ist die IP-Adresse des Servers oder der Maschine, mit dem sich verbunden werden soll. Existieren mehrere Ports, also Netzwerkendpunkte für einen Host, wird zusätzlich der gewünschte angegeben. Der Standardport hat die Nummer 1521. Die SID ist die Kennung für eine bestimmte Datenbankinstanz. Ist ein Service Name als Alias für eine oder mehrere Datenbankinstanzen hinterlegt, kann stattdessen auch dieser angegeben werden. Benutzername und -kennwort sind die notwendigen Informationen zur Identifikation unter welchem Schema die Verbindung hergestellt wird.

Bei erfolgreicher Verbindung gibt die Methode ein Objekt der Klasse Connection zurück, dessen Hauptfunktion die Erstellung eines Cursors ist (siehe Abbildung 3-9). Ein Cursor kann Befehle auf das DBMS ausführen und ist damit das zentrale Glied in der Kommunikation mit der Datenbank. Handelt es sich bei dem Befehl um eine SQL-Abfrage, wird das Ergebnis der Abfrage an den Cursor gebunden. Die Daten können anschließend zeilenweise oder falls der Speicher genügt, in ihrer Gesamtheit mittels der fetch-Methoden abgerufen werden. Es ist also möglich aus einer Programmierumgebung heraus eine Verbindung mit einer Oracle Datenbank herzustellen und per SQL-Befehl Informationen

in die Umgebung zu laden, sowie aus der Umgebung Informationen in ein Datenbankschema zu schreiben (siehe Quelltext 3-1).

Quelltext 3-1: Python-Skript – Auszug Datenbankverbindung mit python-oracledb

```
oracledb.init_oracle_client(lib_dir=r"resources/instantclient_19_9")
      >> Initialisierung des Client
connection = oracledb.connect(
            user="Lennard_Buttgereit",
            password="passwort",
            host="localhost",
            port=1521,
            sid="orcl")
      >> <Connection to Lennard_Buttgereit@localhost:1521>
cursor = connection.cursor()
      >> <Cursor on Lennard_Buttgereit@localhost:1521
cursor.execute("SELECT * FROM fenster")
fenster = cursor.fetchall()
      >> NUMMER       NAME             TYP
      >>   1      Fenster_OG1_78    Drehfluegel
      >>   2      Fenster_EG_2      Kipp
      >>                                                                    […]
```

Für die objektorientieren Erweiterungen des RDBMS existieren die Klassen DBObject, DBObjectType und DBObjectAttribute. Mit ihnen werden sowohl Collection-Typen als auch strukturierte Typen zugänglich. Die Einträge von Kollektionen können mithilfe bekannter Listenmanipulationen gesteuert werden. Bei strukturierten Typen erfolgt dies durch direkten Attributaufruf. Über die vorgestellten Inhalte hinaus beinhaltet das Modul weitere Funktionen, die allerdings im Rahmen der Arbeit nicht benötigt werden.

3.5.3. PY2NEO

Analog zu python-oracledb für Oracle Database besteht in py2neo die Möglichkeit aus einer Python-Umgebung heraus mit einer Neo4j Graphendatenbank zu kommunizieren (Nigel Small, 2021).

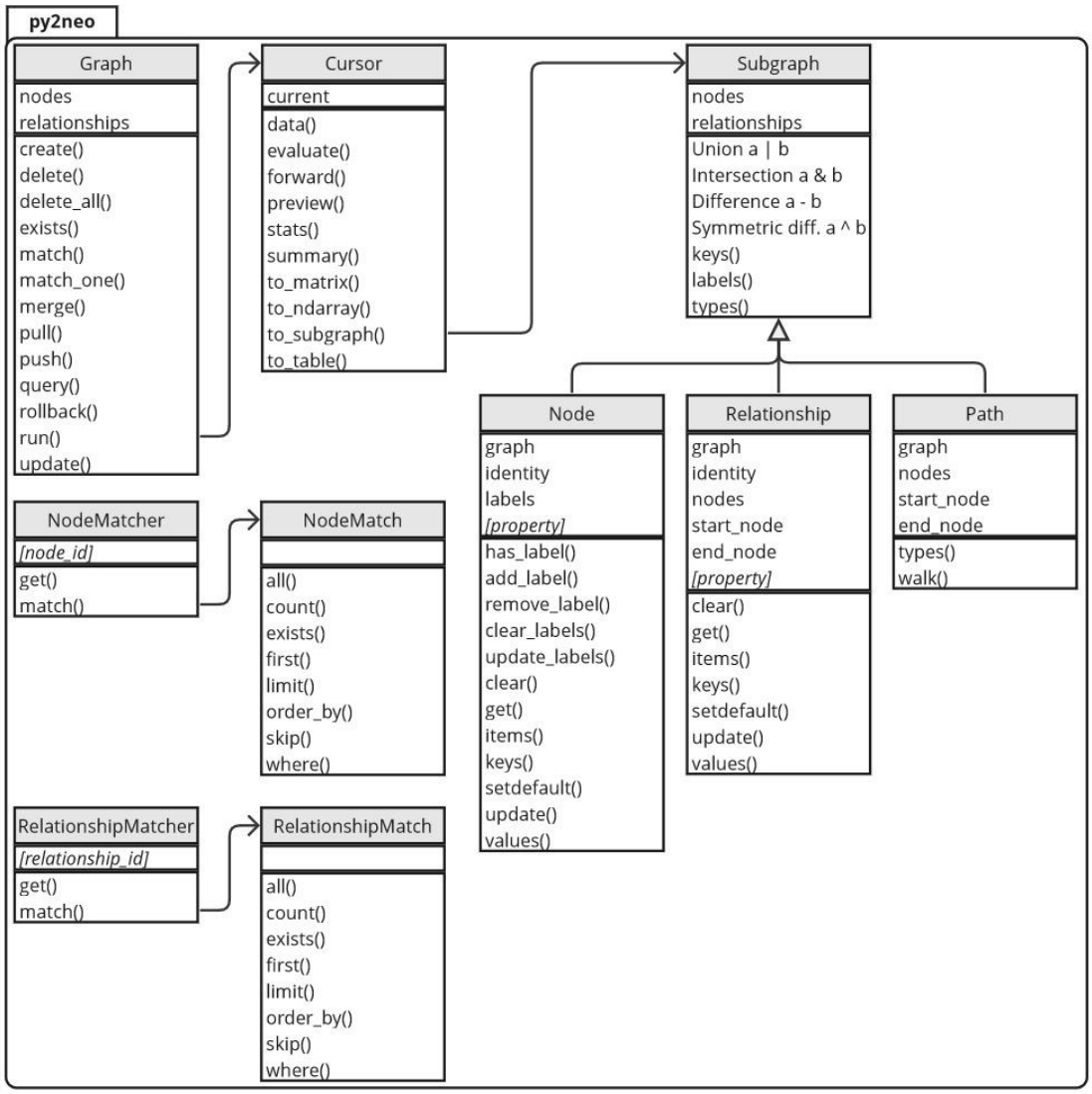


Abbildung 3-10: Auszug py2neo Modul

Wieder beginnt die Interaktion mit einem Verbindungsobjekt, hier von der Klasse Graph (siehe Abbildung 3-10). Zur Instanziierung werden die IP-Adresse, der Port und die Benutzerinformationen übergeben, wenngleich die Parameternamen anders lauten. Die IP-Adresse wird als profile, der Port als name und Benutzername, sowie -kennwort zusammengefasst als auth bezeichnet. Bei erfolgreicher Verbindung werden anschließend über die Attribute nodes und relationships alle Knoten und Kanten des Graphens ausgegeben, jeweils als Liste von Node- bzw. Relationship-Objekten. Mit der Funktion run() werden Queries in der Neo4j-Sprache Cypher ausgeführt. Das Ergebnis eines ausgeführten Cypher-Befehls ist ein Objekt der Klasse Cursor. Die Funktionen des Cursors sind analog zu denen in oracledb. Je nach erwartetem Ergebnis der Abfrage wird zwischen den Datensätzen navigiert oder die Daten werden transformiert zu beispielsweise einem Subgraph-Objekt. Diese Elternklasse umfasst Knoten (Node), Kanten (Relationship) und Pfade (Path). Letzteres ist die Kombination aus mindestens einem Knoten und einer

Kante, wenn der Knoten über die Kante mit sich selbst in Beziehung steht. In der Regel handelt es sich aber um eine alternierende Folge von Knoten und Kanten. Statt Knoten und Kanten über eine selbst zu definierende Cypher-Query abzufragen, besteht in den Klassen NodeMatcher und RelationshipMatcher auch eine automatisierte Abfragemöglichkeit mit einfachen Übergabeparametern. Da Neo4j ein DBMS für Daten in LPG-Form ist, bildet py2neo die Knoten- bzw. Kantenmerkmale auf Objektattribute ab. Die konkreten Merkmalwerte sind durch Attributaufruf zugänglich und manipulierbar.

Die Erstellung von neuen Daten erfolgt entweder über Cypher-Queries oder über die Erstellung von Node- bzw. Relationshipobjekten, die auf die Datenbank gespiegelt werden. Der Standardbefehl hierfür lautet create() auf das Verbindungsobjekt von Typ Graph (siehe Quelltext 3-2).

Quelltext 3-2: Python-Skript – Auszug Verbindung zu Neo4j Datenbank mit py2neo

```
graph = Graph(
            profile="bolt://localhost:7687",
            name="neo4j",
            author=("Lennard_Buttgereit", "B4u1nf0rm4t1k_15t_t0ll"))
      >> Graph(bolt://localhost:7687, name=neo4j)

NUR = Node("Haus", Name="Nuernberger Ei")
OG2 = Node("Geschoss", Name="OG2" Nutzung="CIB")
hatGeschoss = Relationship(haus, "hatGeschoss", geschoss)

graph.create(NUR)
graph.create(OG2)
graph.create(hatGeschoss)

cursor = graph.run("""
      MATCH  p = (Haus{Name:"Nuernberger Ei"})-[hatGeschoss]-(Geschoss)
      RETURN p
""")

subgraph = cursor.to_subgraph()
subgraph.nodes
      >> Node("Haus", Name="Nuernberger Ei")
      >> Node("Geschoss", Name="OG2" Nutzung="CIB")

subgraph.nodes[0].Name
      >> "Nuernberger Ei"
```

In py2neo besteht ein umfangreiches Tool, um Daten sowohl objektbezogen als auch über die Befehlssprache Cypher in einer Neo4j-Datenbank zu verwalten.

3.5.4. IFCOPENSHELL

Ein Teil der Arbeit ist das Überführen von Daten aus einem IFC-SPF in die Datenbankensysteme. Mit oracledb und py2neo können grundsätzlich Daten aus einer Python-Programmierumgebung in das jeweilige System geschrieben werden. In IfcOpenShell haben zudem mehrere Entwickler gemeinsam eine Schnittstelle geschaffen, die EXPRESS-Modelle parst und in Python-Klassen umwandelt (Kijnen & Schultz, 2023). Das Modul erlaubt auch das Parsen des SPFF. Dabei werden die zeilenbasierten Instanzen des STEP Physicial File Format auf EXPRESS-konforme Klasseninstanzen eines Python-Objektmodells zugeordnet.

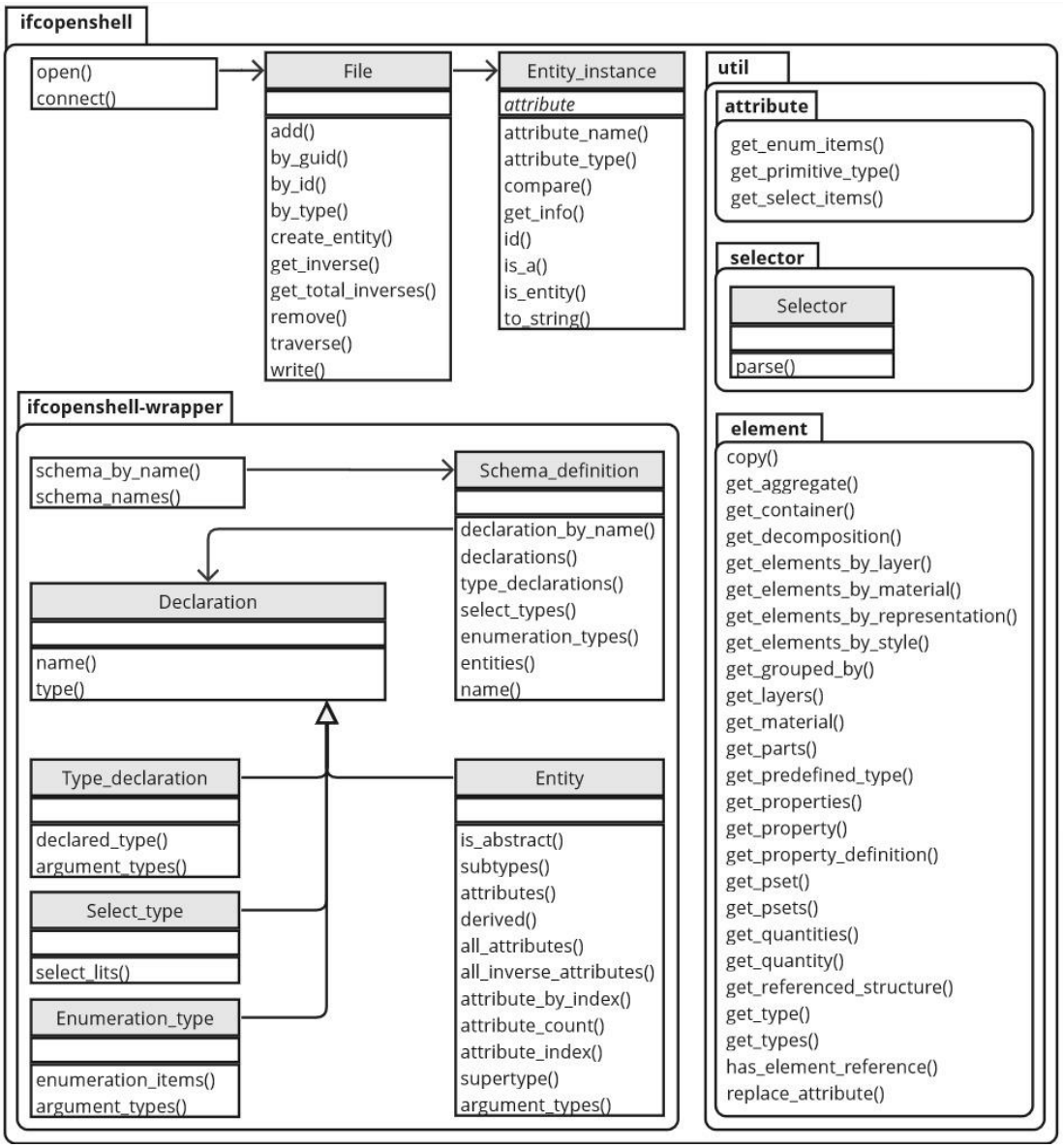


Abbildung 3-11: Auszug ifcopenshell Modul

Die Hauptanwendung von IfcOpenShell ist das Einlesen und Manipulieren eines bestehenden oder die Erstellung eines neuen IFC-SPF. Die wichtigsten Klassen dafür sind File und Entity_instance (siehe Abbildung 3-11). Ein File-Objekt ist der Container für eine beliebige Zahl von Entity_instance-Objekten. Die Existenz letzterer ist jedoch nicht an die ersterer gebunden. Ein bereits bestehendes IFC-SPF wird geöffnet durch die klassenunabhängige Methode open() und die Übergabe des entsprechenden Dateipfades. Ergebnis des Aufrufs ist das File-Objekt, dessen Inhalt über verschiedene Methoden zugänglich ist (Vgl. Quelltext 3-3).

Quelltext 3-3: Python-Skript – Parsen eines Modells mit IfcOpenShell

```
model = ifcopenshell.open("Architektur.ifc")
      >> #1= IfcOrganization($,'Autodesk Revit 2022 (DEU)',$,$,$);
      >> #5= IfcApplication(#1,'2022','Autodesk Revit 2022 (DEU)','Revit');
      >> […]

walls = model.by_type("IfcWall")
      >> #12681=IfcWall('00zNxJ1S91sOivUNwFxl9f',#42,'Basiswand:GK',[…])
      >> #12859=IfcWall('00zNxJ1S91sOivUNwFxl9g',#42,'Basiswand:GK',[…])
      >> […]

instance = model.by_guid('00zNxJ1S91sOivUNwFxl9f')
      >> #12681=IfcWall('00zNxJ1S91sOivUNwFxl9f',#42,'Basiswand:GK',[…])

instance = model.by_id(1)
      >> #1= IFCORGANIZATION($,'Autodesk Revit 2022 (DEU)',$,$,$);
```

Durch Iteration über das File-Objekt, zum Beispiel in Form einer Zählschleife, werden alle Instanzen des Modells ausgegeben. Sollen stattdessen nur Instanzen eines bestimmten Typs, beispielsweise IfcWall, ausgegeben werden, geschieht dies durch den Filter by_type(). Über dessen Ergebnisse lässt sich anschließend wieder iterieren. Um eine bestimmte Instanz auszugeben, wird entweder dessen ID im SPF oder, falls vorhanden, dessen GlobalId genutzt. Jede Instanz, egal wie erhalten, ist ein Objekt der Klasse Entity_instance. Dessen Attributwerte werden ausgegeben durch Aufruf des Attributnamens oder Angabe der Stelle des Attributes. Alternativ beinhaltet die Klasse allgemeingültigere Methoden zum Werteabruf (siehe Quelltext 3-4). Die Methode get_info() gibt zudem ein Dictionary, also eine Sammlung von Schlüssel-Werte-Paaren für das Objekt zurück, das auch Eigenschaften über die EXPRESS-Definition hinaus ausgibt.

Quelltext 3-4: Python-Skript – Parsen einer Instanz mit IfcOpenShell

```
walls = model.by_type("IfcWall")
wall = walls[0]
      >> #12681=IfcWall('00zNxJ1S91sOivUNwFxl9f',#42,'Basiswand:GK',[…])

wall.GlobalId
      >> '00zNxJ1S91sOivUNwFxl9f'
wall.OwnerHistory
      >> #42=IfcOwnerHistory(#39,#5,$,.NOCHANGE.,$,$,$,1676988973)
wall[2]
      >> 'Basiswand:GK'
wall.get_info()
      >> id: 12681
      >> type: 'IfcWall'
      >> GlobalId: '00zNxJ1S91sOivUNwFxl9f'
      >> […]
```

Mit den beiden vorgestellten Grundklassen können alle modellrelevanten Informationen aus einem IFC-SPF extrahiert werden. Dafür ist in der Regel ein substanzielles Wissen des IFC-Metamodells nötig. Einige häufige Anwendungsfälle beim Parsen einer Modelldatei sind daher im Submodul util zusammengefasst (Vgl. Abbildung 3-11). Darunter fällt die Ausgabe der räumlichen Zuordnung mit der Methode get_container(), aber auch die Rückgabe aller zu einem Element zugehörigen PropertySets mit get_psets().

Hinter der intuitiven Handhabung mit konkreten IFC-Daten liegt die Initialisierung des zugrundeliegenden Schemas. Mit dem Submodul ifcopenshell-wrapper liefert IfcOpenShell einen Parser für EXPRESS-Modelle, der IFC-Schemata in Klassen und Objekte der Python-Sprache überträgt. Unterstützt werden aktuell IFC2X3 und IFC4, aber auch zurückgezogene Versionen und solche, die noch in Bearbeitung sind. Dank des flexiblen Kompilierungskonzepts ist es unproblematisch, jede Version verfügbar zu machen, solange sie im EXPRESS-Datenformat veröffentlich wird. Jede Schemaversion ist als Objekt der Klasse Schema_definition durch den klassenunabhängigen schema_by_name()-Aufruf zugänglich. Für dieses Objekt lassen sich anschließend alle getroffenen Definitionen einsehen als Instanzen von Declaration. Darunter fallen Entitäten, aber auch Datentypen, deren Inhalte je nach Fall durch klassenabhängige Methoden zugänglich sind.

Quelltext 3-5: Python-Skript – Parsen eines Schemas mit IfcOpenShell

```
schema = ifcopenshell.ifcopenshell_wrapper.schema_by_name("IFC4")
wall = schema.declaration_by_name("IfcWall")
      >> <entity IfcWall>
wall.all_attributes()
      >> <attribute GlobalId: <type IfcGloballyUniqueId: <string>>>
      >> <attribute OwnerHistory?: <entity IfcOwnerHistory>>
      >> […]
wall.supertype()
      >> <entity IfcBuildingElement>
```

Eine spezifische Deklaration wird durch die declaration_by_name()-Methode unter Angabe dessen Namen abgerufen (siehe Quelltext 3-5). Der Rückgabewert ist ein Objekt einer der vier Subklassen von Declaration. Auf dieses Objekt werden klassenspezifische Methoden angewendet, um benötigte Informationen wie die Attributdefinitionen oder den direkten Supertypen auszugeben. Alternativ zur Auswahl per Name ist es auch möglich, sämtliche Deklarationen eines Schemas über entsprechende Methoden der Schema_definition abzurufen.

4 IMPLEMENTIERUNG

Mit den vorgestellten Ansätzen, kompatiblen DBMS, unterstützenden Anwendungen und Programmierschnittstellen wird nachfolgend der Arbeitsablauf vorgestellt. Es wird aufgezeigt, wie die Komponenten ineinandergreifen und welche Schritte für die jeweilige Modelltransformation nötig sind.

4.1. ORACLE OBJEKTRELATIONALE DATENBANK

Das Endziel des objektrelationalen Ansatzes ist es, Daten eines IFC-SPF in Objekte, gebunden an Relationen einer Oracle Datenbank zu überführen. Dafür muss zuvor eine Schemastruktur erstellt werden, die in der Lage ist, solche Daten sinnvoll zu speichern. Für den Entwurf werden die verschiedenen Modellebenen des Oracle SQL Developer Data Modeler konstruiert. Anschließend wird die Schemadefinition auf eine Datenbankinstanz angewendet und mittels eines Algorithmus mit den Daten eines IFC-SPF befüllt.

4.1.1. DATENBANKSCHEMA

Der Oracle SQL Developer Data Modeler unterstützt die Entwicklung eines Datenbankschemas durch eine Reihe ineinandergreifender Modelle, die zu einem komplexen Entwurf zusammengestellt werden. Dazu zählt das Datentypen- und das logische Modell, sowie beliebig viele relationale und physische Modelle (Vgl. Abbildung 3-8). Jedes Teilmodell kann mitunter aus verschiedenen Komponenten bestehen. Im Nachfolgenden wird vorgestellt, wie die einzelnen Komponenten und Ebenen genutzt werden, um ein möglichst IFC-konformes Datenbankschema zu entwerfen.

Domains

Der Arbeitsablauf sieht zunächst die Erstellung eines Datentypenmodells vor. Das Grundgerüst stellen die primitiven Typen, in Oracle auch logische Typen oder Oracle-Built-Ins genannt. Idealerweise können die primitiven Datentypen in EXPRESS direkt auf jene abgebildet werden (Vgl. Tabelle 4-1).

Tabelle 4-1: Abbildung der EXPRESS Datentypen auf Oracle Datentypen

EXPRESS	ORACLE	Bemerkung
Number	NUMBER(p,s)	Allgemeine Zahl der Präzision (p) und Skala (s)
Real	NUMBER(p,s)	Allgemeine Zahl der Präzision (p) und Skala (s)
Integer	NUMBER(p,0)	Ganzzahl – Allgemeine Zahl mit s=0
Logical	/	Nur als Domain abbildbar
Boolean	/	Nur als Domain abbildbar
String	VARCHAR2(n)	Zeichenkette der Größe n (8-Bit-Zeichenkodierung)
Binary	/	Nur als Domain abbildbar

NUMBER(p,s) ist das allgemeine Zahlenformat in Oracle. Mittels der zwei Variablen Präzision (p) und Skala (s) lassen sich zahlreiche Teilmengen formulieren. Der Wert p gibt die Anzahl von relevanten Dezimalstellen mit einem Maximum von 38 Ziffern an. Die Skala s ist die Definition von signifikanten Stellen nach bzw. bei negativem Wert vor dem Komma. Die Teilmenge Integer lässt sich abbilden, indem die Skala den Wert Null erhält, also keine Nachkommastellen existieren dürfen.

Das allgemeine Zeichenformat in Oracle ist VARCHAR2(n) für reine 8-Bit-Zeichenkodierung. Die Variable n gibt die Größe an, also wieviel Zeichen erlaubt sind bzw. die Maximalanzahl belegter Bytes. Für Oracle 12cR2 liegt die Maximalgröße bei 4000 Bytes (Oracle Corporation, 2016). Für 8-Bit-Kodierung, z.B. ASCII, resultiert draus eine Zeichenkette von maximal 4000 Zeichen. Neben der Definition variabler Zeichenketten unter Angabe einer Maximallänge ist mithilfe des Datentypen CHAR(n) eine Zeichenkette von immer genau der gleichen Länge definierbar. Die EXPRESS-Typen Logical, Boolean und Binary sind nicht ohne weiteres auf Oracle Built-Ins abbildbar. Das Konzept der direkten textuellen Interpretation des Zwei-Zustände-Prinzips existiert in Oracle nicht. Sie können aber in Form einer Domain erstellt werden.

Eine Domain ist die Spezifikation eines logischen Typens in Oracle. Neben der Angabe von Präzision und Skala für Zahlenwerte, sowie der Länge für Zeichenketten gibt es die Optionen Bereich und Werteliste. Der Typ Logical umfasst die Werte False, Unknown und True. Die Domain Logical wird erstellt, indem einem VARCHAR2(7) die Werteliste {'False'; 'Unknown'; 'True'} zugeordnet wird. Analog erfolgt die Definition von Boolean mit den Werten False und True. Binary ist in EXPRESS definiert als eine Binärfolge von variabler oder fester Größe (ISO, 2004). Dieser Datentyp kann abgebildet werden als VARCHAR2(n). Dass nur die Zeichen 0 und 1 erlaubt sind, ist allerdings nicht modellierbar. Insofern ist ein Binary zwar abbildbar, dessen Modellierungsvorschrift aber nicht überprüfbar. Der Datentyp Binary kommt äußerst selten im IFC-Metamodell und auch

konkreten Bauwerksinformationsmodellen vor, weshalb der Fehler als vernachlässigbar angenommen wird.

Ausgehend von den Oracle Built-Ins werden die definierten Typen aus dem IFC-EXPRESS-Modell erstellt. Ein IfcLabel ist nach Definition ein String der Größe 255. Abgebildet auf eine Oracle-Domain ist es ein VARCHAR2(255).

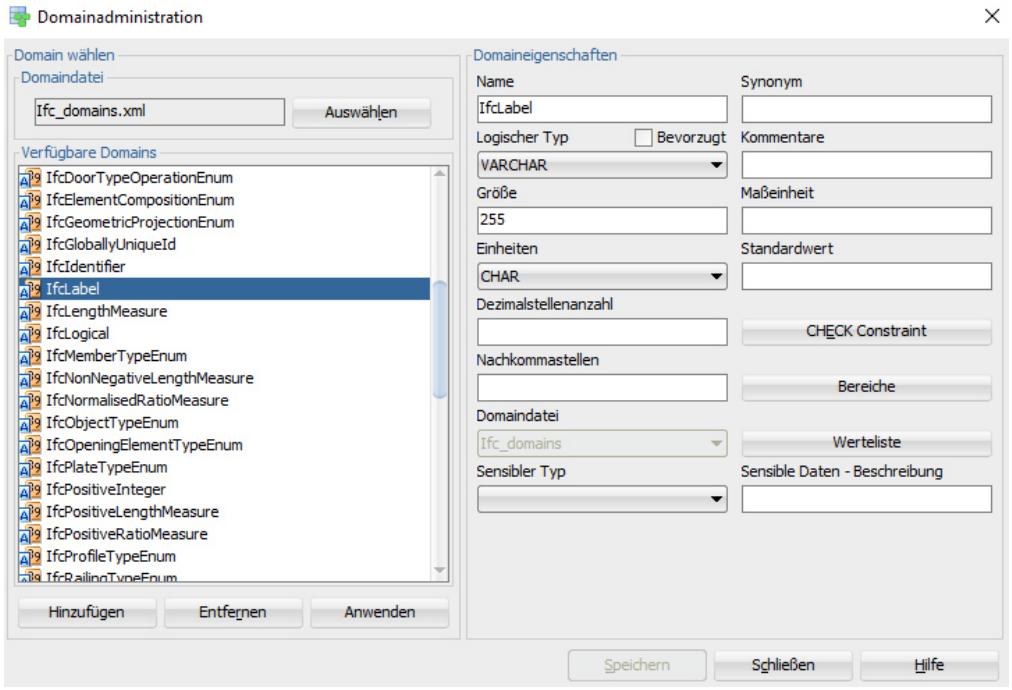


Abbildung 4-1: Domain IfcLabel

Das Vorgehen für viele andere definierte Typen ist analog. Teilweise genügt aber die reine Angabe von Built-In Typ und Größe nicht aus. Der Typ IfcNonNegativeLengthMeasure ist ein IfcLengthMeasure, der nur nichtnegative Werte annehmen darf. Ein IfcLengthMeasure wiederum ist ein primitiver REAL. Die zusätzliche Einschränkung, auch Constraint genannt, wird über die Definition eines Wertebereichs realisiert. Die untere Grenze ist Null und die obere die maximal abbildbare Zahl. Grundsätzlich kann der Datentyp NUMBER eine Präzision von bis zu 38 Stellen aufweisen. Die größte Festkommazahl ist daher eine Folge von 38 Neunen. Darüber hinaus ist die Definition von Gleitkommazahlen mit einem positiven Exponenten von bis zu 125 und einem negativen Exponenten von bis zu 130 zur Basis 10 möglich. Die so entstehende größtmögliche Zahl ist eine Neun exponenziert mit 125 zur Basis 10. Ein IfcNonNegativeLengthmeasure kann also definiert werden als NUMBER mit dem Wertebereich [0; 9E+125]. Das Gegenstück IfcPositiveLengthMeasure, bei dem die Null exkludiert ist, erhält als untere Grenze den kleinsten abbildbaren Wert. Genutzt wird 1E-130, also Eins exponenziert mit -130 zur Basis 10. Nach diesem Vorgehen sind die definierten Typen in IFC auf Domains im

Oracle SQL Developer Data Modeler abbildbar. Für die in IFC zahlreich verwendeten Enumerationen existiert ebenfalls Abhilfe durch Domains. Es handelt sich um nichts anderes als Zeichenfolgen mit definierten Wertelisten. Die erlaubten Werte sind zu übertragen in die Domaindefinition und werden später als Constraint auf ein entsprechendes Attribut gespiegelt. Auf diesem Weg sind fast 60 benötigte Domains entstanden.

Strukturierte Typen

Hauptbestandteil des vorgeschlagenen Ansatzes sind strukturierte Typen. Sie sind gleichbedeutend mit dem Begriff der Klasse aus dem objektorientierten Kontext. Beide werden nachfolgend synonym verwendet. Für den objektrelationalen Ansatz müssen alle relevanten IFC-Entitäten in Oracle-Klassen überführt werden. Ein strukturierter Typ muss mindestens einen Namen und ein Attribut aufweisen. Darüber hinaus kann ein Supertyp angegeben werden, dessen Attribute der Typ erbt. In diesem Fall muss der Typ nicht zwingend eigene Attribute aufweisen. Weiterhin kann dem Typen Instanziierbarkeit zu- oder abgesprochen werden. Ein nicht instanziierbarer Typ wird abstrakte Klasse genannt. Jedem definierten Attribut muss ein Datentyp zugewiesen werden. Möglich sind logische Typen, Domains, Referenzen auf andere strukturierte Typen oder Kollektionstypen. Attribute können obligatorisch oder optional sein. Ein erzeugtes Objekt muss einen Wert für obligatorische Attribute haben, für optionale nicht. Als Klassennamen muss eine Variation der eigentlichen Namen der IFC-Entitäten fungieren. Grund dafür ist die Eindeutigkeit von Namen für alle Datenbankobjekte eines Schemas. Die Namen nach EXPRESS-Definition werden zur besseren Wiedererkennung für die Objekttabellen eingesetzt und sind somit nicht mehr für die strukturierten Typen möglich. Letztere erhalten eine Kombination aus dem Namen und einem vorangestellten Unterstrich (siehe Abbildung 4-2).

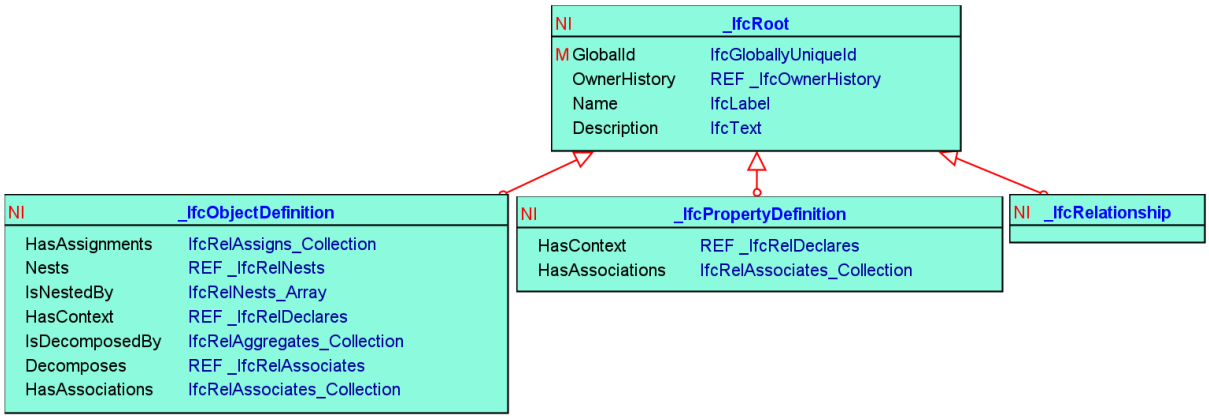


Abbildung 4-2: Strukturierter Typ _IfcRoot und direkte Subtypen

Über die Vererbung hinaus wird eine Beziehung zwischen zwei strukturierten Typen modelliert, indem Typ A ein Attribut vom Typ B erhält (siehe Abbildung 4-3).

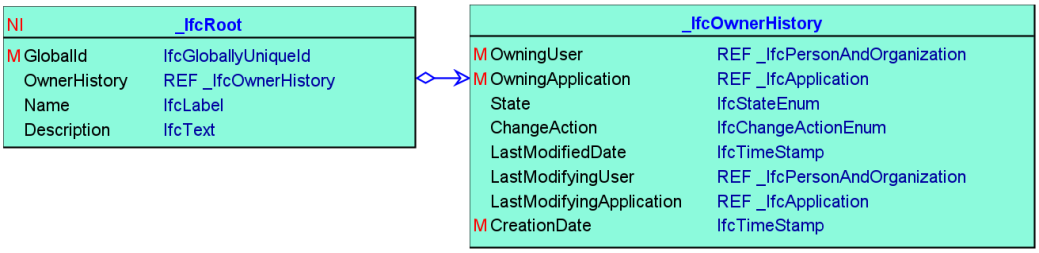


Abbildung 4-3: Strukturierte Typen _IfcRoot und _IfcOwnerHistory

Soll das gesamte Teilmodell Architektur abgebildet werden, müssen auf diesem Weg alle 91 in dem IFC-SPF enthaltenen unterschiedlichen Entitäten modelliert werden (siehe Tabelle 3-4). Jedoch handelt es sich bei den 91 Klassen nur um die instanziierbaren Subtypen. Für ein konsistentes Datenmodell müssen zusätzlich auch alle Supertypen entwickelt werden, wodurch 41 weitere Entitäten nötig sind. Darüber hinaus weisen einige Attribute Objektreferenzen auf wieder neue Typen auf, die ebenfalls zumindest als Hülle angelegt werden. Zum Schluss sind fast 180 strukturierte Typen im Datentypenmodell enthalten. Um einen Überblick über diese Vielzahl zu behalten, ist es möglich Subansichten des Modells zu erzeugen. Als Vorlage für die Strukturierung dient das Schichtenmodell des IFC-Metamodells. Jeder strukturierte Typ wird in einer Subansicht entsprechend des Schemas, in dem die IFC-Entität enthalten ist, angelegt.

Kollektionen

Ein möglicher Datentyp für Attribute von strukturierten Typen sind Kollektionen. Mit Hilfe diesen in Größe begrenzten oder unbegrenzten Mengen sind die EXPRESS-Aggregationen abbildbar. Jede Kollektion darf nur Elemente eines gleichen Datentyps enthalten. Dementsprechend ist für jede Aggregation eines spezifischen Typens eine eigene Kollektion nötig. Zu den möglichen Datentypen zählen die Oracle-Builtins, Domains, strukturierte Typen und wieder Kolletkionen (siehe Abbildung 4-5). Wie auch die strukturierten Typen, muss Jede Kollektion als Teil des Datenbankschemas einen individuellen Namen erhalten. Im Rahmen der Arbeit setzt sich der Name einer Kollektion aus dem zugrundeliegenden Datentyp mit dem Suffix „_Collection“ zusammen (siehe Abbildung 4-4).

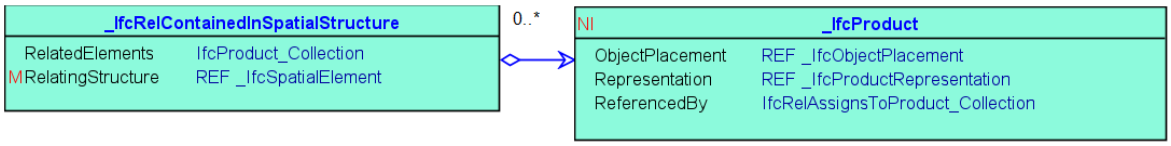


Abbildung 4-4: Strukturierte Typen _IfcRelContainedInSpatialStructure und _IfcProduct

Diese Vorlage gilt für alle nicht in Größe beschränkten Aggregationen. Sogenannte Arrays, Aggregationen mit oberer Schranke, werden durch eine Konvention „_Array_n“ umgesetzt, wobei n die Maximale Anzahl von Elementen ist.

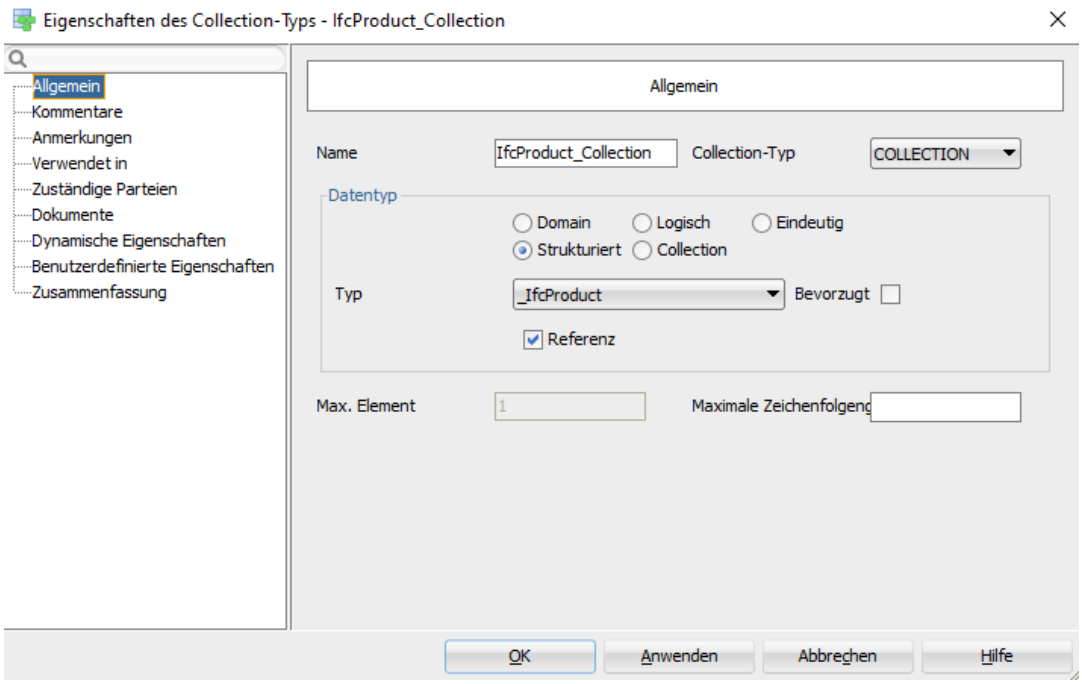


Abbildung 4-5: Kollektionstyp IfcProduct_Collection

In objektrelationalen Datenbanken ist es nicht möglich, eine untere Grenze für Kollektionen anzugeben. Das liegt daran, dass Kollektionen als verschachtelte Tabellen verwaltet werden und eine Mindestanzahl von Tupeln mengentheoretisch nicht definierbar ist. Für die Attribut-Datentypen der angelegten strukturierten Typen sind über 60 Kollektionstypen im Datentypenmodell entstanden.

Selektionstypen

Der letzte verbleibende Datentyp im EXPRSS-Modell ist der Selektionstyp. Ähnlich wie bei Enumerationen wird eine Auswahlliste erstellt. Bei den Elementen der Liste handelt es sich allerdings nicht um feste Werte, sondern Datentypen. Es kann sich beispielsweise um eine Selektion aus IfcObjectDefinition und IfcProductDefinition handeln. Dieser er-

möglicht die Referenzierung aller Klassen unter IfcRoot, außer denen unter IfcRelationship (Vgl. Abbildung 4-2). Alternativ ist auch eine Auswahl von definierten Typen, beispielsweise IfcReal und IfcText möglich bei kontextabhängigen Attributwerten. In SQL besteht keine Möglichkeit, Selektionstypen direkt abzubilden. Es sind allerdings einige Hilfskonstruktionen denkbar. Option Eins ist, den Selektionstypen in Form eines strukturierten Typen als Supertyp für die Auswahl von Typen anzulegen. Da SQL-Definitionen im Gegensatz zu EXPRESS aber keine Mehrfacherbung erlauben, funktioniert dieser Ansatz nicht. Option Zwei ist, die Selektion wieder als strukturierten Typen anzulegen, die Elemente der Auswahl aber als Attribute zu definieren (siehe Abbildung 4-6).

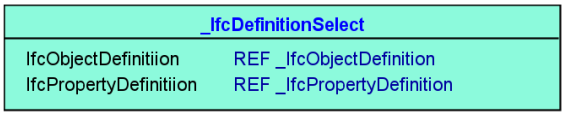


Abbildung 4-6: Selektionstyp _IfcDefinitionSelect

Je nachdem, welcher Typ tatsächlich vorliegt, wird das eine oder das andere Attribut mit einem Wert befüllt. Es wird dadurch gewährleistet, dass nur eine erlaubte Referenz angegeben ist, ohne in das vorhandene Klassenmodell einzugreifen. Die Selektionstypen befinden sich in einer eigenen Subansicht „Select_types“. Damit ist das Datentypenmodell komplementiert.

Logisches Modell

Als nächstes werden Entitäten im logischen Modell angelegt. Der in Oracle verwendete Begriff Entität soll die abstrakte Definition einer Relation darstellen. Nach dem vorgeschlagenen Ansatz werden nur die nicht abstrakten Klassen des IFC-Metamodells als Objektrelationen im logischen Modell entwickelt. Bei der Abbildung eines strukturierten Typens auf eine Entität werden sämtliche Attribute des Vererbungsbaums als Attribute der Entität übernommen. Zusätzlich gibt es das private Attribut des Object Identifiers (OID), die Referenz für jedes Objekt. Wie für Relationen erforderlich wird ein Primärschlüsselattribut identifiziert. Für die Klassen unter IfcRoot wird die GlobalId verwendet. Die restlichen können unter Angabe der Oid identifiziert werden (Vgl. Abbildung 4-7).

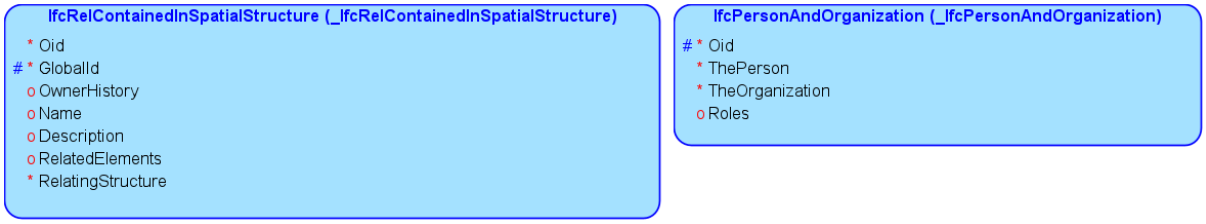


Abbildung 4-7: Entitäten IfcRelContainedInSpatialStructure und IfcPersonAndOrganization

Auf diesem Weg erhält das logische Modell die 91 Entitäten des Teilmodells Architektur, die die konkreten Klassen im IFC-Metamodell wiederspiegeln. Dabei ist jede Entität unmittelbar an den entsprechenden strukturierten Typen geknüpft. Änderungen des Typen werden also direkt auf die Entität übernommen.

Relationales Modell

In der Theorie soll das logische Modell einmal alle konkreten Entitäten des IFC-Metamodells enthalten. Anschließend können dann für ein projektspezifisches Datenbankschema exakt die Entitäten in das relationale Modell übernommen werden, die für das Projekt notwendig sind. Die Vorgabe für die Liste an Entitäten kann beispielsweise aus einer Auftraggeber-Informations-Anforderung (AIA) entstehen. Im Rahmen der Arbeit werden alle 91 Entitäten überführt. Während die GlobalId als offen zugängliches Primärschlüsselattribut übernommen wird, muss die Oid als Schlüssel erneut zugewiesen werden (siehe Abbildung 4-8)

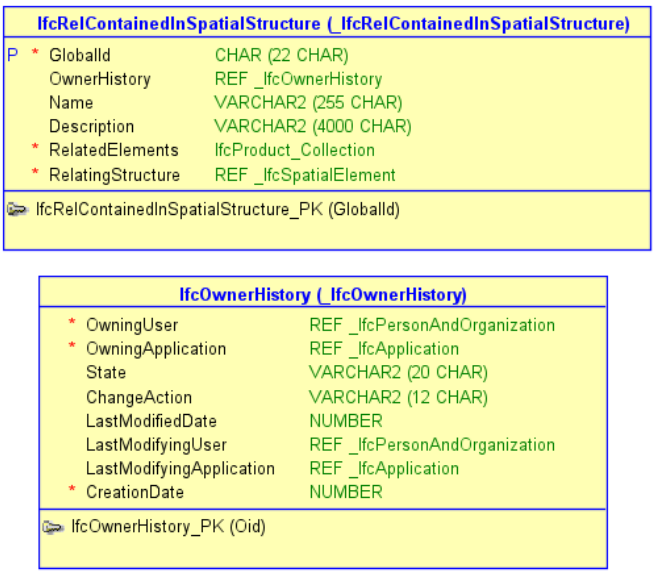


Abbildung 4-8: Tabellen IfcRelContainedInSpatialsStructure und IfcOwnerHistory

Mit Überführung der Entitäten werden die Domains wieder zurück auf Oracle-Built-Ins abgebildet, so zum Beispiel IfcGloballyUniqueId auf CHAR(22). Das Anlegen der Domains hat dennoch Bedeutung, da die Änderung einer solchen auch automatisch alle Attributdefinitionen aktualisiert. Alternativ wäre nur die fehleranfällige manuelle Überprüfung und ggfs. Anpassung aller Attribute möglich.

Physisches Modell

Aus dem relationalen Modell heraus lassen sich Befehle in einer Data Definition Language (DDL) generieren, mit denen konkrete Datenbankinstanzen erstellt werden. Für das Teilmodell Architektur wird ein SQL-Skript generiert, dass auf die Oracle Database 12.2 anwendbar ist. Es besteht aus einer Vielzahl von Befehlen, von denen das Gros auf die strukturierten Typen zurückzuführen ist. Sie werden durch den Create-Type-Befehl in Oracle erstellt (siehe Quelltext 4-1). Das Kennwort „Not final“ symbolisiert, dass der aktuelle Typ ein Supertyp anderer sein kann. Die Vererbung findet statt durch die Angabe des Schlüssels „Under“ in der Definition eines Subtypens mit Verweis auf den Supertypen. Mit „Not instantiable“ wird ein Typ als abstrakt deklariert.

Quelltext 4-1: SQL-Skript – Typ _IfcRoot

```
CREATE OR REPLACE TYPE "_IfcRoot" AS OBJECT (
    globalid     CHAR(22 CHAR),
    ownerhistory REF "_IfcOwnerHistory",
    name         VARCHAR2(255 CHAR),
    description  VARCHAR2(4000 CHAR)
) NOT FINAL NOT INSTANTIABLE;
```

Objektrelationen werden wie normale Relationen mittels des Create-Table-Befehls erstellt, wobei die Of-Type-Klausel genutzt wird, um den zugrundeliegenden Typen anzugeben (siehe Quelltext 4-2). An dieser Stelle erfolgt auch bereits ein Teil der Restriktionen für bestimmte Attribute. Obligation wird beispielsweise durch die Kennung „Not null“ indiziert.

Quelltext 4-2: SQL-Skript – Objekttabelle IfcOwnerHistory

```
CREATE TABLE ifcownerhistory OF "_IfcOwnerHistory" (
    owninguser NOT NULL,
    owningapplication NOT NULL,
    creationdate NOT NULL
);
```

Enumerationen können nicht in der Tabellengenerierung überprüft werden. Stattdessen werden sie nachträglich als Einschränkung mittels Add-Check-Klausel als Teil einer Tabellenänderung angelegt (siehe Quelltext 4-3). Dabei wird das entsprechende Attribut benannt und anschließend die Liste an validen Werten.

Quelltext 4-3: SQL-Skript – Enumeration PredefinedType

```
ALTER TABLE ifccolumn
    ADD CHECK (
      predefinedtype IN (
            'COLUMN',
            'NOTDEFINED',
            'PILASTER',
            'USERDEFINED'
      )
);
```

Noch nicht behandelt ist die Instanziierung von Kollektionen. Wie auch strukturierte Typen werden sie durch den Create-Type-Befehl erstellt, hier aber ohne die As-Object-Klausel. Zudem werden keine Attribute angegeben, sondern nur die Information, dass es sich um eine verschachtelte Tabelle von bestimmtem Typ handelt (siehe Quelltext 4-4).

Quelltext 4-4: SQL-Skript – Kollektion IfcProduct_Collection

```
CREATE OR REPLACE TYPE ifcproduct_collection IS
    TABLE OF REF "_IfcProduct";
```

Der Kollektionstyp ist daraufhin ein möglicher Datentyp für das Attribut eines strukturierten Typen. Wird anschließend dieser strukturierte Typ als Grundlage für eine Relation gewählt, erfolgt eine Übersetzung der Kollektion auf eine verschachtelte Tabelle.

Quelltext 4-5: SQL-Skript – Verschachtelte Tabelle RelatedElements

```
CREATE OR REPLACE TYPE "_IfcRelContainedInSpatialStructure" UNDER
"_IfcRelationship" (
    relatedelements   ifcproduct_collection,
    relatingstructure REF "_IfcSpatialElement"
) FINAL;

CREATE TABLE ifcrelcontainedinspatialstructure
      OF "_IfcRelContainedInSpatialStructure" (
    globalid NOT NULL,
    [ … ]
)
NESTED TABLE relatedelements
    STORE AS ifcrelcontainedinspatialstructure_relatedelements;
```

Es entsteht für jedes Collection einer Relation eine eigene verschachtelte Tabelle. Der Oracle-Algorithmus hinter dem SQL-Skript setzt als Namen für die Tabelle automatisch den Namen des Attributes ein. Wenn eine Kollektion auf einem übergeordneten Supertypen definiert wird, erhalten allerdings alle Subtypen den selben Attributnamen. Es

entstünde eine Vielzahl gleichnamiger Datenbankobjekte. Da das nicht erlaubt ist, müssen die Namen manuell bzw. durch ein zusätzliches Skript angepasst werden. Als Konvention wird vorgeschlagen, für jede verschachtelte Tabelle den Attributnamen zu wählen, aber den Ursprungstabellennamen als Prefix mit Unterstrich zu nutzen (siehe Quelltext 4-5, Zeile 13). Auf die verschachtelte Tabelle kann nicht direkt zugegriffen werden. Die enthaltenen Elemente sind aber über das entsprechende Attribut eines Objektes zugänglich.

Datenbankschema

Die SQL-Befehle des Skriptes sind durch einen Algorithmus in einer solchen Reihenfolge, dass zunächst alle benötigten strukturierten Typen und Kollektionen und anschließend die Tabellen generiert werden. Unter den Objektdatentypen ist außerdem eine solche Abfolge zu beachten, dass für einen strukturierten Typen benötigte andere Typen vor dessen Generierung bereits zumindest als Hülle existieren. Das Endergebnis sind über 170 strukturierte Typen, über 60 Kollektionstypen und die bekannten 91 Tabellen (Vgl. Abbildung 4-9).

Abbildung 4-9: Spalten der Tabelle IfcOwnerHistory

IFCOWNERHISTORY

Spalten | Daten | Model | Constraints | Zugriffsberechtigungen | Statistiken | Trigger | Flashback | Abhängigkeiten | Details | Partitionen | Indizes | SQL

Aktionen…

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	OWNINGUSER	_IfcPersonAndOrganization	Yes	(null)	1	(null)
2	OWNINGAPPLICATION	_IfcApplication	Yes	(null)	2	(null)
3	STATE	VARCHAR2(25 CHAR)	Yes	(null)	3	(null)
4	CHANGEACTION	VARCHAR2(12 CHAR)	Yes	(null)	4	(null)
5	LASTMODIFIEDDATE	NUMBER	Yes	(null)	5	(null)
6	LASTMODIFYINGUSER	_IfcPersonAndOrganization	Yes	(null)	6	(null)
7	LASTMODIFYINGAPPLICATION	_IfcApplication	Yes	(null)	7	(null)
8	CREATIONDATE	NUMBER	Yes	(null)	8	(null)

4.1.2. DATENBEFÜLLUNG

Der nächste Schritt ist das Befüllen des leeren Datenbankschemas mit Daten aus der SPFF-Serialisierung des Teilmodells Architektur. Vor dem Erstellen eines Algorithmus, der diese Aufgabe automatisiert ausführt, ist es sinnvoll die einzelnen Befehle zu verstehen. Als erstes werden die Objekte angelegt, wobei die Attribute simplen Datentyps direkt mit den entsprechenden Werten belegt werden. Die Erstellung folgt dem gleichen Muster wie bei der Erstellung eines Tupels in einer relationalen Datenbank (Vgl. Quelltext 4-6).

Quelltext 4-6: SQL-Skript - Objekte vom Typ _IfcPerson und _IfcOrganization

```
INSERT INTO ifcperson (identification, familyname, givenname)
VALUES ('4718286', 'Buttgereit', 'Lennard');

INSERT INTO ifcorganization (identification, name)
VALUES ('TUD', 'Technische Universität Dresden');
```

Tatsächlich wird durch den Befehl aber nicht wie sonst ein Tupel, sondern ein referenzierbares Objekt mit den angegebenen Werten als Attributen angelegt. Die Referenz wird anschließend direkt in die Objekttabelle geschrieben und kann genutzt werden, um die Beziehungen unter den Objekten zu modellieren. Da nicht direkt auf die Oid eines Objektes zugegriffen werden kann, erfolgt die Attribuierung durch die REF-Methode und eine Select-Where-Klausel (siehe Quelltext 4-7).

Quelltext 4-7: SQL-Skript - Objekt vom Typ _IfcPersonAndOrganizationon

```
INSERT INTO ifcpersonandorganization (theperson, theorganization)
    SELECT REF(p), REF(o)
    FROM ifcperson p, ifcorganization o
    WHERE p.identification = '4718286'
      AND o.identification = 'TUD';
```

Auch das Anlegen von Kollektionen gestalten sich sehr vertraut. Es genügt, den Namen des Kollektionstypen anzugeben und die Werte einzuspeisen (siehe Quelltext 4-8).

Quelltext 4-8: SQL-Skript – Objekt vom Typ IfcRelContainedInSpatialContainment

```
INSERT INTO ifcrelcontainedinspatialstructure (relatedelements)
    SELECT ifcelement_collection(
        REF(w1), REF(w2), REF(c1), REF(c2)
    )
    FROM ifcwall w1, ifcwall w2, ifccolumn c1, ifccolumn c2
    WHERE w1.globalid = '00zNxJ1S91sOivUNwFxl9f'
      AND w2.globalid = '00zNxJ1S91sOivUNwFxl9g'
      AND c1.globalid = '00zNxJ1S91sOivUNwFxlEE'
      AND c2.globalid = '00zNxJ1S91sOivUNwFxlE8';
```

Nachdem die grundlegenden Befehle bekannt sind, müssen sie nun geschickt in einen Algorithmus eingebaut werden, der die Daten aus einem IFC-SPF einliest und in ausführbare SQL-Befehle umwandelt. Aus einer Python-Programmierumgebung heraus ist es möglich mit dem Modul IfcOpenShell alle Informationen aus dem IFC-SPF greifbar zu machen. Nachdem die Daten in SQL-Befehle umgewandelt sind, sind sie mithilfe des python-oracledb-Moduls auf das Datenbankschema anzuwenden (Vgl. Abbildung 4-10).

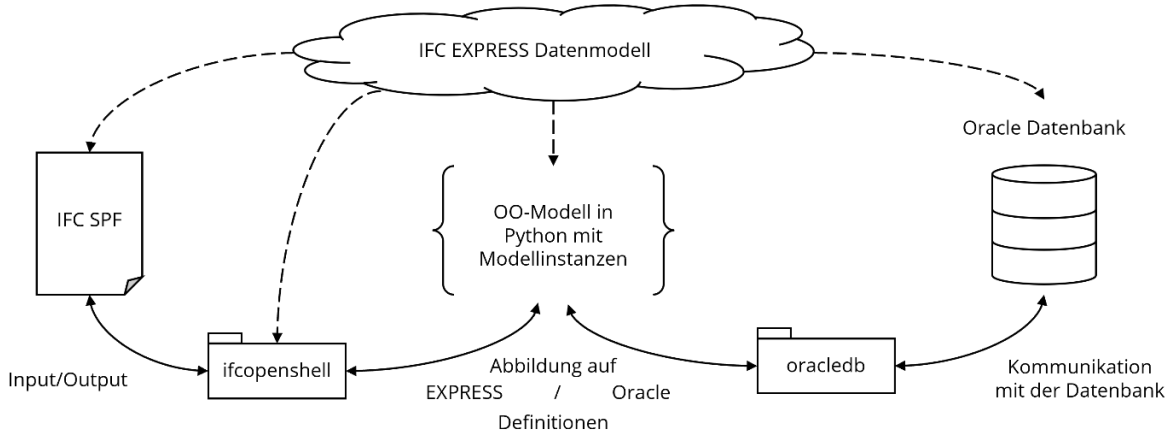


Abbildung 4-10: Systemarchitektur IFC-SPF zu Oracle Database Algorithmus

Beim Befüllen der Schemastruktur ist die Reihenfolge ebenso wichtig wie bei der Erstellung des Schemas. Zunächst sollen alle Objekte mit ihren Attributen primitiven Datentyps, also weder Objektreferenz noch Kollektion, erstellt werden. Dabei ist es sinnvoll, die gerade angelegten Objekte auch im temporären Programmspeicher zu behalten für eine spätere Zuordnung. Allerdings weisen nicht alle IFC-Entitäten überhaupt solche Attribute auf, geschweige denn solche, die später zur eindeutigen Erkennung genutzt werden können. In einem zweiten Schritt müssen genau solche Instanzen des Bauwerksinformationsmodells gefunden und mit mindestens einem Wert einer Referenz bzw. Kollektion belegt werden. Auf diesem Weg werden zunächst alle Instanzen aus dem Modell auch als Objekte mit minimaler Attribuierung in der Datenbank angelegt. Über SQL-Update-Befehle werden anschließend die Beziehungen zwischen den Objekten mittels Angabe der Referenzen modelliert werden. Zuletzt erfolgen die Kollektionen von Referenzen.

4.2. NEO4J GRAPHENDATENBANK

Für den graphenbasierten Ansatz entfällt der Entwurf einer Schemastruktur. Es kann direkt eine Konvertierung der Instanzen nach EXPRESS-Definition des IFC-Metamodells in Knoten und Kanten erfolgen. Nach dem vorgeschlagenen LPG-Prinzip ist jede Instanz ein eigenständiger Knoten mit dem Label der entsprechenden IFC-Entität, sowie all seiner Supertypen. Attribute von simplem Datentyp werden als Eigenschaften des Knotens angelegt. Beziehungsattribute hingegen sind als Kanten modelliert. Das Python-Modul py2neo erlaubt für die Erstellung des Graphen Methoden, die über das Ausführen von Cypher-Queries weit hinausgehen. Mit den Klassen Node und Relationship des Moduls werden Knoten und Kanten zunächst im Programmspeicher angelegt und mittels eines Commits auf die Datenbank gespiegelt (Vgl. Abbildung 4-11).

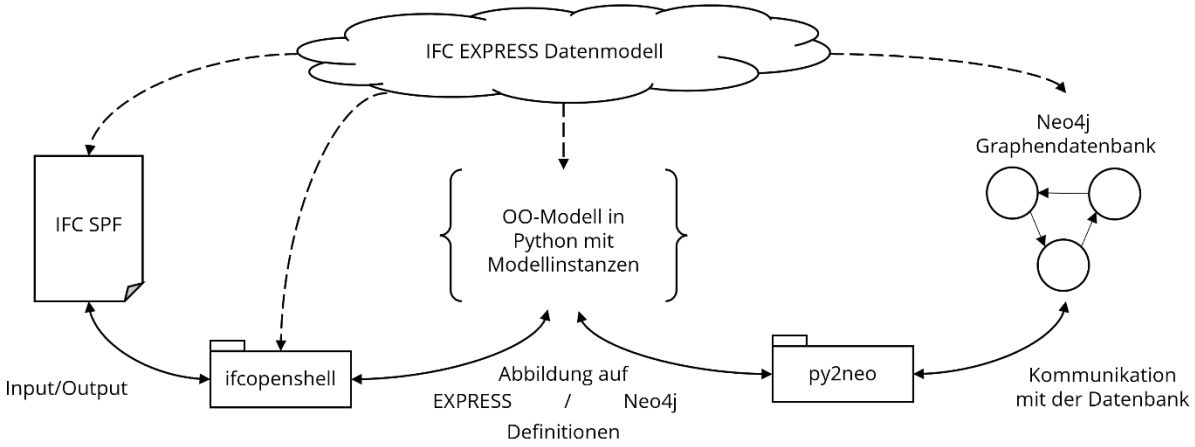


Abbildung 4-11: Arbeitsablauf für Neo4j Graphendatenbank

Wie auch für den objektrelationalen Ansatz spielt die Reihenfolge bei der Transformation eine große Rolle. Im Fall des Graphen müssen zuerst alle Knoten vorhanden sein, um dann Beziehungen modellieren zu können. Jeder Knoten wird als py2neo-Node-Objekt angelegt. Als Labels sind der direkte IFC-Entitätsname, sowie alle Namen seiner Supertypen angegeben. Das erlaubt später eine übergeordnete Ausgabe von Knoten auf abstrakter Klassenebene. Für die Knoteneigenschaften wird eine Schlüssel-Wert-Menge erstellt, die die Attribute einfachen Datentyps enthält. Sowohl für die Labels als auch die Knoteneigenschaften stammen die Informationen aus der Verwendung des IfcOpenShell-Moduls. Durch Iteration über die Instanzen des IFC-SPF sind alle anzulegenden Knoten bekannt. Für jede Entität werden durch Parsen des IFC-Metamodells mit dem ifcopenshell-wrapper-Modul die für die Labels benötigten Supertypen erhalten. Anschließend erfolgt eine Iteration über die Attribute der Instanz. Solche, die weder von aggregiertem Datentyp noch eine Objektreferenz sind, erhalten einen Eintrag in der Schlüssel-Wert-Menge. Aggregierte Datentypen werden in Neo4j aktuell nicht als Datentyp für Knoteneigenschaften unterstützt. Stattdessen kommt eine Transformation zu einer textuellen Repräsentation enthaltener Elemente zum Einsatz.

Ist die Erstellung aller Instanzen als Knoten erfolgt, werden die Objektattribute, also die Beziehungen behandelt. Sie werden als Kante unter Angabe der zu verbindenden Knoten und des Attributnamens erstellt. Dabei erfolgt wieder zunächst die Erzeugung interner py2neo-Relationship-Objekte, die danach auf die Datenbank gespiegelt werden. Mit der vorgestellten Implementierung sind keine Cypher-Queries zur Erstellung des Modells nötig.

5 VALIDIERUNG

Als Validierung für die beiden Modelltransformationen des IFC-SPF werden folgende Kriterien vorgeschlagen:

1. Das Modell enthält genauso viele Instanzen wie das SPF
2. Das Modell enthält pro Entität genauso viele Instanzen wie das SPF
3. Instanzen des Modells und des SPF weisen die gleichen Attribute auf (Stichprobe)
4. Eine Rücktransformation in das SPFF ist möglich
5. Eine Rücktransformation erzeugt ein gleiches SPF

Das Teilmodell Architektur des CIB-Modells enthält 25420 Instanzen. Die in das jeweilige Datenbankschema geschriebenen Datensätze können mittels Anfragen ermittelt werden. Im Fall der Oracle Datenbank werden alle Zeilen über alle Objektrelationen gezählt (siehe Quelltext 5-1).

Quelltext 5-1: SQL-Skript – Anzahl von Objekten

```
SELECT SUM(num_rows)
FROM user_object_tables;

SELECT table_name, num_rows
FROM user_object_tables;
```

Für den graphenbasierten Ansatz wurden alle Instanzen in Knoten überführt. Die benötigte Cypher-Query zählt daher alle Knoten des Graphen (siehe Quelltext 5-2).

Quelltext 5-2: Cypher-Skript – Anzahl von Knoten

```
MATCH (n) RETURN count(*)

MATCH (n)
RETURN distinct labels(n), count(*)
```

Ergebnis der Anfragen ist, dass in beiden Datenbankschemata die 25420 Einträge existieren. In einem nächsten Schritt soll überprüft werden, inwiefern auch die Zuordnung der Instanzen auf strukturierte Typen beziehungsweise Label erfolgreich war. Die Originaldaten können mittels des Python-Moduls IfcOpenShell erhalten werden. Dafür werden zunächst die unterschiedlichen Entitäten im Modell und anschließend für jede Entität die Menge an Instanzen ermittelt (siehe Quelltext 5-3).

Quelltext 5-3: Python-Skript – Anzahl von Instanzen je Entität

```
model = ifcopenshell.open("Architektur.ifc")
entities_in_model = set([element.is_a() for element in model])
instances_per_entity = {
      entity: len(model.by_type(entity) for entity in entities_in_model
}
[print(entity, number) for entity, number in instances_per_entity.items()]
```

Die erhaltenen Schlüssel-Wert-Paare aus (Entität: Instanzen_Anzahl) werden verglichen mit den Datensätzen der beiden Datenbankmodelle. In Oracle erfolgt eine Auflistung der Zeilen je Objekttabelle (siehe Quelltext 5-1, Z. 4f.). Die Cypher-Anfrage für Neo4j ermittelt zunächst alle unterschiedlichen Label und dann die Anzahl von Knoten mit diesem Label (siehe Quelltext 5-3, Z.3f.). Das Ergebnis ist auch hier positiv: Die Anzahl von Zeilen je Tabelle und Knoten je Label stimmt für das Teilmodell Architektur überein mit der Anzahl von Instanzen je Entität.

Nachdem die grundsätzliche Existenz der Datensätze bestätigt werden konnte, soll die Frage geklärt werden, ob die Datensätze auch korrekt sind, also inhaltlich mit denen des IFC-SPF übereinstimmen. Beispielhaft wird dafür eine Instanz von IfcRelContainedInSpatialStructure untersucht. Der Umfang wird auf die Instanz selbst und direkt referenzierte Instanzen festgelegt (siehe Quelltext 5-4).

Quelltext 5-4: IFC-SPF – Auszug Teilmodell Architektur

```
#86258= IFCRELCONTAINEDINSPATIALSTRUCTURE(
   '1jPCssxb5C1RSM_YLIx$zl',#42,$,$,(#12681),#163
);

#42= IFCOWNERHISTORY(
   #39,#5,$,.NOCHANGE.,$,$,$,1676988973
);
#12681= IFCWALL(
   '00zNxJ1S91sOivUNwFxl9f',#42,'Basiswand:GK 100:2543869',$,
   'Basiswand:GK 100',#12669,#12678,'2543869',.NOTDEFINED.
);
#163= IFCBUILDINGSTOREY(
   '2lQC93K9nCKfy1iFn3r2ym',#42,'OG 1 - OK FFB',$,
   'Ebene:Ebene OK FFB',#162,$,'OG 1 - OK FFB',.ELEMENT.,3.95000000000007
);
```

Das entsprechende Objekt in der Oracle Datenbank ist mittels eines Select-Befehls unter Angabe der GlobalId zugänglich. Da es sich bei einigen Attributen um Objektreferenzen handelt, sind deren Werte nicht direkt überprüfbar (siehe Abbildung 5-1).

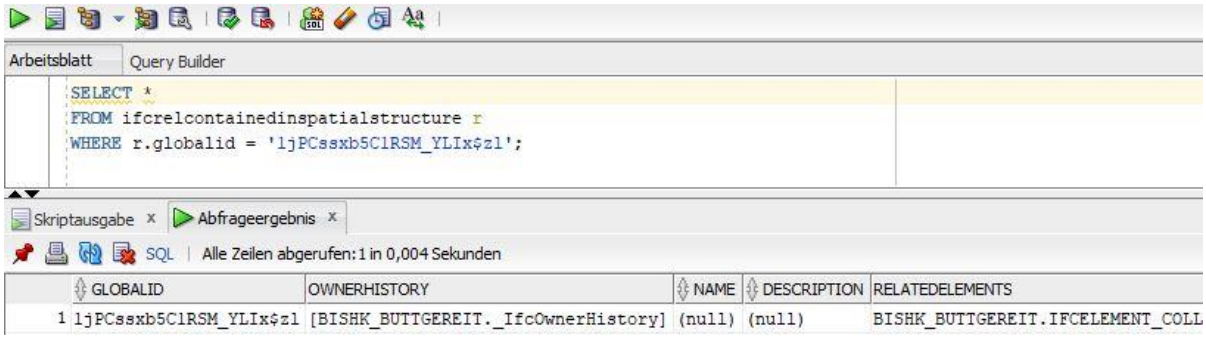


Abbildung 5-1: Oracle SQL Developer – Auszug IfcRelContainedInSpatialStructure

Stattdessen können aber die Attribute der referenzierten Objekte analysiert werden. Dazu gehört beispielsweise der Zeitstempel der Erstellung unter dem Attribut OwnerHistory, die GlobalIds der zugeordneten Elemente und die GlobalId der zuordnenden räumlichen Struktur (siehe Quelltext 5-5).

Quelltext 5-5: SQL-Skript – IfcRelContainedInSpatialStructure

```
/*Der Zeitstempel der Erstellung*/
SELECT r.ownerhistory.creationdate
FROM ifcrelcontainedinspatialstructure r
WHERE r.globalid = '1jPCssxb5C1RSM_YLIx$zl';
      >> 1676988973

/*Die GlobalIds aller zugeordneten Element*/
SELECT re.column_value.GlobalId AS GlobalId
FROM ifcrelcontainedinspatialstructure r, table (r.relatedelements) re
WHERE r.globalid = '1jPCssxb5C1RSM_YLIx$zl';
      >> ('00zNxJ1S91sOivUNwFxl9f')

/*Die GlobalId der betreffenden räumlichen Struktur*/
SELECT r.relatingstructure.globalid
FROM ifcrelcontainedinspatialstructure r
WHERE r.globalid = '1jPCssxb5C1RSM_YLIx$zl';
      >> '2lQC93K9nCKfy1iFn3r2ym'
```

Analog wird der Neo4j-Graph auf die selbe GlobalId für das Label IfcRelContainedInSpatialStructure abgefragt. Dabei können die mit Kanten verbundenen Knoten durch eine Pfad-Query ausgegeben werden (siehe Quelltext 5-6).

Quelltext 5-6: Cypher-Skript – IfcRelContainedInSpatialStructure

```
MATCH
p=(r:IfcRelContainedInSpatialStructure{GlobalId:'1jPCssxb5C1RSM_YLIx$zl'})
   -[]->()
RETURN p
```

Das Ergebnis sind vier Knoten und fünf Kanten. Bei den Knoten handelt es sich um die Instanzen für IfcRelContainedInSpatialStructure, IfcOwnerHistory, IfcWall und IfcBuildingStrorey. Die Kanten sind die Beziehungsattribute für jene Instanzen, mittels derer sie verbunden sind. Ausgeführt in der Neo4j-Browser-Anwendung kann der Pfad auch visuell eingesehen werden (siehe Abbildung 5-2).

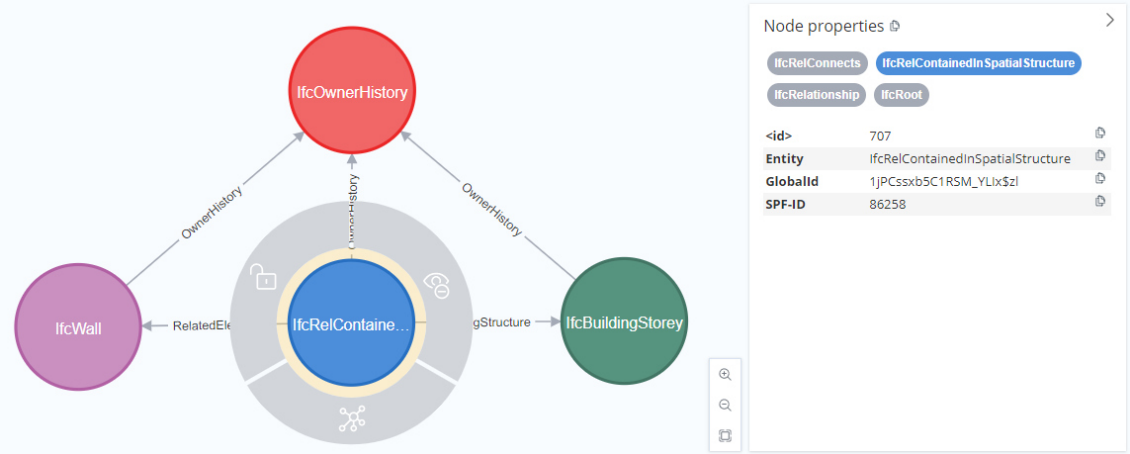


Abbildung 5-2: Neo4j Browser – IfcRelContainedInSpatialStructure

Das Ergebnis beider Stichproben ist positiv. Die in dem IFC-SPF-enthaltenen Informationen werden auch über Anfragen auf das jeweilige DBMS erhalten. Da es sich nur um ausgewählte Instanzen handelt, kann allerdings nicht mit voller Sicherheit davon ausgegangen werden, dass auch alle anderen Modellinhalte vollends korrekt abgebildet wurden. Eine Möglichkeit, um die Vollständigkeit sicherzustellen, ist die Rücktransformation in ein IFC-SPF mit anschließendem Vergleich aller Instanzen. Alternativ kann auch ein neutrales Datenformat wie das Comma-Separated-Values-Format (CSV) genutzt werden. Hier ist die Analyse dann aber abhängig von der gewählten Struktur und den Inhalten. Die Wahl einer IFC-Serialisierung ist daher auf der sicheren Seite. Im Rahmen der Arbeit konnte diese abschließende Analyse aufgrund des hohen technischen Aufwands nicht erfolgen. Der Vergleich der Metadaten und die Stichprobe sollen daher genügen, um die Modelle zumindest in Ansätzen zu validieren.

6 BEWERTUNG UND AUSBLICK

Sowohl das graphenbasierte als auch das objektrelationale Datenbankmodell sind grundlegend geeignet, um IFC-konforme Bauwerksinformationsmodelle dateigelöst zu verwalten. Während der objektrelationale Ansatz durch eine klare Strukturierung, die standardisierte Abfragesprache und etablierte Arbeitsabläufe besticht, weist der graphenbasierte Ansatz eine höhere Flexibilität und Skalierbarkeit auf.

6.1. OBJEKTRELATIONALER ANSATZ

Entgegen der anfänglichen Annahme, dass das rein relationale Datenbankmodell ebenfalls geeignet ist, muss festgestellt werden, dass dies nur unter Inkaufnahme fehlender IFC-Konformität möglich ist. Die objektrelationale Abbildung ist zwar eine gängige Methode zur relationalen Verwaltung objektorientierter Daten, macht jedoch von Vereinfachungen Gebrauch, die den Wiedererkennungswert des zugrundeliegenden Klassenmodells stark einschränken. Stattdessen wird vorgeschlagen, die Datenspeicherung objektgebunden durchzuführen, das Datenmanagement aber auf Relationen aufzubauen. Möglich ist dies durch das Prinzip der Objektrelation, deren Aufgabe es ist, gespeicherte Objekte auf eine Tupelstruktur abzubilden. Das erlaubt die Datenmanipulation mit etablierten Mitteln und der standardisierten Befehlssprache SQL bei IFC-naher Datenverwaltung.

Das entwickelte Datenbankschema ist in der lage, einen gewissen Umfang des IFC-Metamodells abzubilden, um ein Beispielbauwerkinformationsmodell zu verwalten. Dabei sind einige Teile des Modells leichter transformierbar als andere. Die **definierten Typen** der EXPRESS-Notation sind unproblematisch überführbar in Datentypen eines DBMS. Im Fall der Oracle Database und dem SQL Data Modeler können die Abbildungen als Domains sogar gespeichert und wiederverwendet werden.

Enumerationen können ebenfalls mit einfachen Mitteln, hier Domains mit definierten Wertelisten modelliert werden. Für aggregierte Datentypen ist die Überführung nicht trivial. Für jede Art von enthaltenen Elementen bedarf es eines eigens erstellten Kollektionstypens. Ein allgemeiner Aggregationstyp existiert nicht. Zudem unterstützt zumindest Oracle Database nur entweder in Größe begrenzte oder unbegrenzte, sortierte Aggregationen. In EXPRESS hingegen können mit den Datentypen Bag und Set auch unsortierte und sogar von Duplikaten ausgeschlossene Mengen modelliert werden.

Das Prinzip der **Selektionstypen** ist außerhalb der EXPRESS-Notation eine seltene Modellierungsart. SQL-Datenbanken unterstützen sie in der Regel nicht. Im Rahmen der Arbeit wird allerdings eine Lösung vorgestellt, mit Hilfe derer auch solche Typen abbildbar sind. Die Modellierung weicht insofern vom IFC-Metamodell ab, als dass die Selekti-

onstpyen ebenfalls als strukturierte Typen konstruiert werden. Deren Attribute stellen die verschiedenen Elemente der Selektionsliste dar. Nur jenes Attribut erfährt einen Wert, dessen Datentyp auch der tatsächlich gewählte ist.

Es verbleiben nur noch die **Entitäten** des EXPRESS-Modells. Sie werden auf SQL-Klassen, strukturierte Typen genannt, abgebildet. Jedes Attribut der Entität wird als Attribut der Klasse angelegt. Als Datentypen fungieren die zuvor eingeführten Abbildungen von definierten Typen, Enumerationen, Selektionstypen und Aggregationen.

Die Untersuchung ergibt, dass das IFC-Metamodell grundsätzlich auch durch die objektorientierten Erweiterungen von SQL abgebildet werden kann. Datensätze werden objektgebunden gespeichert, aber über die Maske von Relationen verwaltet. Es genügt dabei, die Relationen zu bilden, deren korrelierenden Klassen auch in konkreten IFC-Modellinstanzen vorhanden sind. Damit wird das zu verwaltende Modell von den 776 Entitäten auf 653 konkrete Entitäten in IFC4 reduziert. Eine Analyse des gängigen Modellierungsumfangs zeigt auf, dass von den 653 Entitäten nur ein Bruchteil tatsächlich Verwendung findet. Durch Restriktionen auf Seiten der Softwarehersteller, aber auch durch den bereichsabhängig reduzierten Modellinhalt weist beispielsweise eines der untersuchten Bauwerksinformationsmodelle nur 132 unterschiedliche Entitäten auf. Nach dem verfolgten Ansatz entspricht das 132 Relationen des Datenbankschemas. Der Umfang wird als gut verwaltbar eingeschätzt. Der objektrelationale Datenbankansatz ist damit grundsätzlich geeignet zur dateigelösten Verwaltung von Bauwerksinformationsmodellen auf Basis des IFC-Metamodells.

Dabei sind noch nicht sämtliche Modellierungsmöglichkeiten ausgeschöpft. Die definierten strukturierten Typen weisen aktuell nur Attribute auf, können aber auch Methoden abbilden. Dadurch ist es möglich, die in EXPRESS explizit definierten Regeln und Funktionen umzusetzen. Dazu gehört beispielsweise die automatisierte Schlussfolgerung der Dimension eines kartesischen Punktes durch die Menge an Punkten (siehe Quelltext 6-1).

Quelltext 6-1: EXPRESS-Skript – Definition Entität IfcCartesianPoint

```
ENTITY IfcCartesianPoint
 SUBTYPE OF (IfcPoint);
  Coordinates : LIST [1:3] OF IfcLengthMeasure;
 DERIVE
  Dim : IfcDimensionCount := HIINDEX(Coordinates);
 WHERE
  CP2Dor3D : HIINDEX(Coordinates) >= 2;
END_ENTITY;
```

Für den Modellierungsumfang der Arbeit ist das Datenbankschema per Hand mit Hilfe des SQL Data Modelers entstanden. Nötig war dies, um zum Schluss SQL-Anweisungen

zu erhalten, deren Anwendung auf eine Datenbankinstanz das gewünschte Schema erstellt. Stattdessen ist es aber auch denkbar, einen Algorithmus zu entwickeln, der die SQL-Befehle basierend auf Parsen des IFC-Metamodells erstellt. Die benötigten technischen Komponenten sind mit den vorgestellten Python-Modulen verfügbar. Insbesondere zum Erstellen der strukturierten Typen ist eine Automatisierung sinnvoll. In Zukunft soll durch sie das gesamte IFC-Klassenmodell, also alle 776 Entitäten, abgebildet werden. Das händische Übertragen der Definitionen aus IFC-Dokumentationen ist ineffizient und prädestiniert für Modellierungsfehler.

6.2. GRAPHBASIERTER ANSATZ

Der Entwurf eines Schemas entfällt für den LPG- Ansatz. Stattdessen ist das Ergebnis ein flexibles Python-Skript, das in der Lage ist, jedwede Information aus einem IFC-SPF in Knoten und Kanten zu transformieren. Der Algorithmus behandelt die einzelnen Aspekte des Metamodells dabei unterschiedlich. Definierte Typen werden auf die ihnen zugrundeliegenden primitiven EXPRESS-Datentypen reduziert. Ein IfcLabel ist demnach ein String, der direkt auf den in Neo4j gleichnamigen primitiven Typen abgebildet wird (Vgl. Quelltext 6-2).

Quelltext 6-2: EXPRESS-Skript – Definierter Typ IfcLabel

```
1  TYPE IfcLabel = STRING (255);
2  END_TYPE;
```

Analog sind sämtliche Enumerationen Strings mit bestimmtem Attributwert. Aggregierte Datentypen können hingegen nicht trivial abgebildet werden. Neo4j unterstützt, wie andere LPG-DBMS-Anbieter auch, keine mengenartigen Knoten- und Kanteneigenschaften. Im Rahmen der Arbeit wird eine Hilfskonstruktion vorgeschlagen, bei der die Menge an Werten in eine String-basierte Darstellung transformiert wird. Eine Liste [0; 1; 2] ist anschließend ein String „(0, 1, 2)“. Mit dieser Vereinfachung wird der Attributwert als Eigenschaft angelegt und ist per Query auch zugänglich. Lediglich die einzelnen Elemente können nicht direkt ausgegeben werden. Der Selektionstyp spielt im Rahmen der konkreten Modellinstantransformation eine untergeordnete Rolle. Ist das zugrundeliegende IFC-SPF mit dem Metamodell konform, so ist auch der Wert eines Attributes vom Datentyp Selektion korrekt. Die Übertragung in den Graphen erfolgt nach beschriebenem Muster. Letztlich verbleibt der Datentyp Entität. Er wird im Gegensatz zu den anderen Typen nicht als direkte Eigenschaft, sondern in Tripelform modelliert. Zwei Knoten werden durch eine Kante vom Label des verbindenden Attributes miteinander in Relation gesetzt.

Im Gegensatz zum objektrelationalen Ansatz stellt die Transformation eines anderen IFC-SPFs in eine Graphendatenbank keinen Mehraufwand dar. Es muss nicht der Modellierungsinhalt überprüft und gegebenenfalls das Datenbankschema erweitert werden. Alle EXPRESS-Definitionen, egal in welcher Version, werden nach dem vorgeschlagenen Muster auf die Graphendatenbank abgebildet. Der Ansatz ist daher deutlich flexibler in der Anwendung und kann durch die Abfragesprache Cypher für Neo4j ebenso nach modellierten Daten abgefragt werden, wie der objektrelationale mit SQL.

Der Vorteil des graphenbasierten Ansatzes ist aber gleichzeitig auch eine Schwäche. Nach aktuellem Vorgehen kann in der Theorie jede Information, egal ob IFC-konform oder nicht, in die Datenbank geschrieben werden. Dazu zählen Labels von Graphelementen, die direkten Knoten- bzw. Kanteneigenschaften, sowie Tripelformulierungen, die nicht zwangsläufig mit dem IFC-Metamodell überinstimmen müssen. Ein konsistentes Datenbankschema benötigt dringend die Definition entsprechender Restriktionen. Der DBMS-Anbieter Neo4j erlaubt durch Nutzer angelegte Einschränkungen auf Elemente bestimmter Labels. Möglich ist beispielsweise die Definition, dass die GlobalId von Knoten des Labels IfcRoot wie auch im Metamodell eindeutig sein muss (siehe Quelltext 6-3).

Quelltext 6-3: Cypher-Skript – Definition GlobalId Uniqueness und Not Null Constraint

```
CREATE CONSTRAINT unique_globalid
FOR (element:IfcRoot) REQUIRE element.globalid IS UNIQUE
```

Auch, dass bestimmte Attribute einer Entität obligatorisch sind, ist übertragbar auf die Labeldefinition von Neo4j-Knoten (Quelltext 6-4).

Quelltext 6-4: Cypher-Skript – Definition GlobalId Not Null Constraint

```
CREATE CONSTRAINT mandatory_globalid
FOR (element:IfcRoot) REQUIRE element.globalid IS NOT NULL
```

Nach Definition der Constraints ist die Flexibilität des graphenbasierten Ansatzes nur genau so stark eingegrenzt, dass die willkürliche Dateneinspeisung reduziert ist. Das Hinzufügen von neuen Knoten, auch solcher, deren Entität zuvor nicht vertreten war, ist hingegen weiterhin möglich.

6.3. AUSBLICK

Die Arbeit stellt einen Beitrag auf dem Weg weg von dateibasiertem, hin zu datenbankbasiertem Datenmanagement im Bauwesen dar. Grundlage und Kern der digitalen Arbeitsweise ist das Bauwerksinformationsmodell. Der dateibasierte Austausch des Mo-

dells in Form eines IFC-SPF hat in der Vergangenheit an Effizienz wünschen lassen und häufig zu redundaten Dateien geführt. Zahlreiche Vorkehrungen sind beispielsweise im Rahmen der CDE getroffen worden, um die Nachteile so gut wie möglich zu reduzieren. Das Ziel von atomarer Datenhaltung kann der dateibasierte Ansatz allerdings nach wie vor nicht realisieren. Ohne zusätzliche Vorkehrungen oder Software ist es nicht möglich, einzelne Elemente auszugeben, geschweige denn zu ändern. Die Ansätze zur Transformation und Serialisierung des IFC-Metamodells auf Datenbankmodelle können hier Abhilfe schaffen. Dank feingranularer Datenhaltung und genau zu diesem Zweck entwickelten, standardisierten Abfrage- und Befehlssprachen sind einzelne Elemente manipulierbar und referenzierbar.

Darauf aufbauend bedarf es nun weiterer Forschung beispielsweise zu den Themen Versionsmanagement und Eigentümerschaft von sowohl Modellen als auch Elementen. Im dateibasierten Austausch wird die Versionierung von Modellen in der Regel durch das projektinterne Teilen der Datei festgehalten. Entweder fungiert eine Namenskonvention oder eine CDE-interne Funktion zum Festhalten der Version. Für den dateilosen Ansatz kann das Konzept durch eine umfangreiche Anzahl von Mechanismen gängiger DBMS umgesetzt werden. Dazu zählt das Transaktionsprinzip und die Zeiterfassung von Datensätzen, sowie die bereits lang etablierte Versionierung. Dabei beschränken sich die Funktinen der Management-Systeme nicht nur auf ganzheitliche Dateien, respektive Modelle, sondern auch auf einzelne Datensätze, also Elemente. Die Informationen des DBMS können durch konsistente Verwendung des Revision-Control-Concepts in IFC mit der Entität IfcOwnerHistory im Zentrum außerdem auch Teil des Modells selbst werden.

Ebenfalls zu untersuchen ist, inwiefern BIM-Prozesse auf Grundlage einer Datenbank ausgeführt werden können. In den Abfragen auf DBMS bestehen beispielsweise neue Möglichkeiten zum BIM Model Checking. Insbesondere die Überprüfung übergebener Modellinhalte kann mittels definierter Queries realisiert werden. Der Export in eine Datei zur Verwendung in einer Software wäre demnach obsolet. Zudem ist es möglich, die Ergebnisse des BIM Model Checking ebenfalls Teil des Bauwerksinformationsmodells werden zu lassen. Die Modellierung der Daten hingegen wird weiterhin in entsprechenden Anwendungen stattfinden. Denkbar sind aber Schnittstellen über den Import und Export von IFC-SPFs hinaus, bei denen die Anwendung direkt mit der Datenbank kommuniziert. Entweder in Echtzeit oder regelmäßigen Commits könnten die Informationen als Datensätze im Schema angelegt werden.

Es ist klar, dass ein datenbankbasiertes BIM aktuell noch weit von einer praktischen Umsetzung entfernt ist. Ein wichtiger Schritt ist aber, über die langfristige Verwaltung von Bauwerkinformationsmodellen zu diskutieren. Hier wird klar, dass der dateibasierte Ansatz früher oder später zu Herausforderungen führt. Namensräume, Dokumentbezeichnungen und die allgemeine Strukturierung der Dateien sind projektabhängig. Festgehalten werden alle Konventionen idealerweise in den Auftraggeber-Informations-

Anforderungen, wodurch sie nachvollziehbar werden. Die Verwaltung jener bei einer Vielzahl von Projekten mit mitunter unterschiedlichen Konventionen erfordert allerdings viel Zeit und Aufwand.

Ein IFC-basiertes Bauwerksinformationsmodell auf Grundlage einer datenbanktechnologischen Anwendung verspricht Abhilfe. Dank des ISO-standardisierten IFC-Metamodells sind Informationen projektübergreifend in einer gleichen Struktur, die zudem gut und ausführlich dokumentiert ist. Spezifische Daten können aus einer Datenbank direkt ausgelesen werden, ohne Modellierungs- oder andere BIM-Software nutzen zu müssen. Für die langfristige, nachhaltige Verwaltung von Modellen ist dieser Ansatz daher überaus vorteilhaft.

Der nächste Schritt in der Weiterentwicklung der Datenbankansätze muss es sein, die Interoperabilität mit den noch gängigen dateibasierten Serialisierungen des IFC-Metamodells zu verbessern. In erster Linie bedarf es dafür flexibler Konvertierungsmöglichkeiten von IFC-SPF-Instanzen zu Datensätzen einer Datenbank und zurück. Mithilfe solcher Technologie ist es denkbar, dass sich die Bauindustrie langsam, aber stetig in ein dateiloses 21. Jahrhundert bewegt.

7 LITERATURVERZEICHNIS

Amann, J., Esser, S., Krijnen, T., Abualdenien, J., Preidel, C. & Borrmann, A. (2022). BIM-Programmierschnittstellen. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *VDI-Buch Ser. Building Information Modeling: Technologische Grundlagen und Industrielle Praxis* (2. Aufl., S. 263–290). Springer Fachmedien Wiesbaden GmbH. https://doi.org/10.1007/978-3-658-33361-4_13

Autodesk Inc. (Hrsg.). (2022). *Revit IFC-Handbuch 2.0.*

Autodesk Inc. (2023). *Revit* (Version 2023) [Computer software]. https://www.autodesk.de/products/revit/overview?term=1-YEAR&tab=subscription

Beetz, J., van Berlo, L., Laat, R. & van den Helm, P. (2010). Bimserver.org - an Open Source IFC model server.

BMI & BMV (Hrsg.). (September 2021). *Masterplan BIM für Bundesbauten*. Berlin, Bonn.

Bock, B. S. & Eder, F. (2022). *IfcSQL* (Version 0.1) [Computer software]. https://github.com/IfcSharp/IfcSQL

Borrmann, A., Beetz, J., Liebich, T. & Muhič, S. (2022). Industry Foundation Classes - Ein herstellerunabhängiges Datenmodell für den gesamten Lebenszyklus eines Bauwerks. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *VDI-Buch Ser. Building Information Modeling: Technologische Grundlagen und Industrielle Praxis* (2. Aufl., S. 95–146). Springer Fachmedien Wiesbaden GmbH. https://doi.org/10.1007/978-3-658-33361-4_6

Borrmann, A., König, M., Koch, C. & Beetz, J. (2022). Die BIM-Methode im Überblick. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *VDI-Buch Ser. Building Information Modeling: Technologische Grundlagen und Industrielle Praxis* (2. Aufl., S. 1–31). Springer Fachmedien Wiesbaden GmbH. https://doi.org/10.1007/978-3-658-33361-4_1

buildingSMART. (2020a, 6. Februar). *IFC4 ADD2 TC1*. https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/

buildingSMART. (2020b, 6. Februar). *IFC4 ADD2 TC1 RV1.2*. https://standards.buildingsmart.org/MVD/RELEASE/IFC4/ADD2_TC1/RV1_2/HTML/

buildingSMART. (2022a, 31. Mai). *IFC Schema Specifications*. https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/

buildingSMART. (2022b, 22. August). *Software Certification*. https://www.buildingsmart.org/compliance/software-certification/

buildingSMART. (2023, 8. Mai). *IFC4.3 ADD1*. https://ifc43-docs.standards.buildingsmart.org/

DIN (2019a-04). *Gemeinsame Datenumgebungen (CDE) für BIM-Projekte - Funktionen und offener Datenaustausch zwischen Plattformen unterschiedlicher Hersteller: Module und Funktionen einer Gemeinsamen Datenumgebung; mit_digitalem_Anhang* (DIN SPEC 91391-1). Berlin. Beuth Verlag GmbH.

DIN (2019b-08). *Organisation und Digitalisierung von Informationen zu Bauwerken und Ingenieurleistungen, einschließlich Bauwerksinformationsmodellierung (BIM) - Informationsmanagement mit BIM: Teil 1: Begriffe und Grundsätze* (DIN EN ISO 19650-1). Berlin. Beuth Verlag GmbH.

DIN (2021-11). *Industry Foundation Classes (IFC) für den Datenaustausch in der Bauwirtschaft und im Anlagenmanagement: Teil 1: Datenschema* (DIN EN ISO 16739-1). Berlin. Beuth Verlag GmbH.

Fuchs, S. (2015). *Erschließung domänenübergreifender Informationsräume mit Multimodellen: = Access of cross-domain information spaces using multi-models*. Zugl.: Dresden, Techn. Univ., Fak. Bauingenieurwesen, Diss., 2015. *Berichte des Instituts für Bauinformatik: Heft 11*. Inst. für Bauinformatik, Fak. Bauingenieurwesen, TU Dresden.

Geisler, F. (2014). *Datenbanken: Grundlagen und Design* (5th ed.). mitp. https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6014703

Gröger, G., Kolbe, T. H., Nagel, C. & Häfele, K. H. (2012). *OGC city geography markup language (CityGML) encoding standard.*

Grren, A., Junghanns, M., Kiessling, M., Lindaaker, T., Plantikow, S. & Selmer, P. (2018). *openCypher: New Directions in Property Graph Querying*. https://dbs.uni-leipzig.de/file/opencypher.pdf

Ismail, A. (2023). *IFC WebServer [Computer software]*. https://ifcwebserver.org/

Ismail, A., Nahar, A. & Scherer, R. (2017). *Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard.*

ISO (1999-12). *Informationstechnik - Datenbanksprachen - SQL: Teil 1: Rahmenwerk (SQL/Rahmenwerk)* (ISO/IEC 9075-1:1999). Berlin. Beuth Verlag GmbH.

ISO (2004-11). *Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch: Teil 11: Beschreibungsmethoden: Handbuch der Modellierungssprache EXPRESS* (ISO 10303-11). Berlin. Beuth Verlag GmbH.

ISO (2007-10). *Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch: Teil 28: Implementierungsmethoden: XML Darsellungen von EXPRESS Schemata und Daten unter Verwendung von XML Schemata* (ISO 10303-28). Berlin. Beuth Verlag GmbH.

ISO (2016a-03). *Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch: Teil 21: Implementierungsmethoden: Klartext-Kodierung der Austauschstruktur* (ISO 10303-21). Berlin. Beuth Verlag GmbH.

ISO (2016b-12). *Informationstechnik - Datenbanksprachen - SQL: Teil 1: Rahmenwerk* (ISO/IEC 9075-1). Berlin. Beuth Verlag GmbH.

Joseph Jabin, Johannes Dimyadi & Robert Amor (2020). Classifying IFC Entities by Their Relative Importance for Accurate Interoperability Measurement. In *CIB W78, 37th International Conference*.

Kijnen, T. & Schultz, R. (2023). *IfcOpenShell* (Version 0.7.0) [Computer software]. https://ifcopenshell.org/

Kiviniemi, A., Fischer, M. & Bazjanac, V. *Integration of multiple product models: IFC model servers as a potential solution.*

Krijnen, T., van Berlo, L., Tauscher, H., Beetz, J., Brouwer, J., Dimyadi, J., Holm, K. V., Bjørkhaug, L., Lindeque, R., Laat, R. de & Kay, L. (2023). *BIMserver: The open source BIMserver platform* (Version 1.5.184) [Computer software]. https://github.com/opensourceBIM/BIMserver

LandXML.org. (2014). *LandXML*. http://www.landxml.org/

Li, H., Liu, H., Liu, Y. & Wang, Y. *An Object-Relational IFC Storage Model Based on Oracle Database.* https://doi.org/10.5194/isprs-archives-XLI-B2-625-2016

Neo4j, I. (2023). *Neo4j Enterprise Edition* (Version 4.4.19) [Computer software]. https://neo4j.com/download-center/#community

Nigel Small. (2021). *py2neo* (Version 2021.2.3) [Computer software]. neo4j. https://neo4j.com/developer/python/#py2neo-lib

Oracle Corporation. (2016). *Oracle Database* (Version 12cR2) [Computer software]. https://docs.oracle.com/database/122/

Oracle Corporation. (2022a). *SQL Developer* (Version 22.2.1) [Computer software]. https://www.oracle.com/database/sqldeveloper/

Oracle Corporation. (2022b). *SQL Developer Data Modeler* (Version 22.2) [Computer software]. https://www.oracle.com/de/database/sqldeveloper/technologies/sql-data-modeler/

Oracle Corporation. (2023a). *Instant Client for Microsoft Windows* (Version 21.9) [Computer software]. https://www.oracle.com/de/database/technologies/instant-client.html

Oracle Corporation. (2023b). *python-oracledb* (Version 1.2.2) [Computer software]. https://oracle.github.io/python-oracledb/

Pauwels, P. & Terkaj, W. (2019). *ifcOWL*. https://standards.buildingsmart.org/IFC/DEV/IFC4/ADD2_TC1/OWL/index.html

Preidel, C., Borrmann, A., Exner, H. & König, M. (2022). Common Data Environment. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *VDI-Buch Ser. Building Information Modeling: Technologische Grundlagen und Industrielle Praxis* (2. Aufl., S. 335–351). Springer Fachmedien Wiesbaden GmbH. https://doi.org/10.1007/978-3-658-33361-4_16

Saake, G. (2018). *Datenbanken: Konzepte und Sprachen* (6. Auflage). mitp Verlags. https://learning.oreilly.com/library/view/-/9783958457782/?ar

Schill, A. & Springer, T. (2012). *Verteilte Systeme: Grundlagen und Basistechnologien* (2. Aufl.). *eXamen.press*. Springer Vieweg. https://doi.org/10.1007/978-3-642-25796-4

Statistisches Bundesamt. (2022, 25. August). *Entwicklung der Beschäftigung in Deutschland (Jahresdaten)*. https://service.destatis.de/DE/vgr-monitor-deutschland/beschaeftigung.html

van Berlo, L., Krijnen, T., Tauscher, H., Liebich, T., van Kranenburg, A. & Paasiala, P. (2021). Future of the Industry Foundation Classes: towards IFC 5. In *Proceedings of the 38th International Conference of CIB W78, Luxembourg, 13-15 October*.

Vossen, G. (2008). *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme* (5., überarb. und erw. Aufl.). Oldenbourg.

W3C. (2014). *RDF*. https://www.w3.org/RDF/

W3C. (2023, 18. April). *OWL*. https://www.w3.org/OWL/

Yadava, H. (2007). *The Berkeley DB book*. *Expert's voice in open source*. Apress; Safari Books Online. https://learning.oreilly.com/library/view/-/9781590596722/?ar

Zelnick, B. A., Mathew, J. & Saleem, S. T. (2017, 5. September). *Oracle Corporation.*

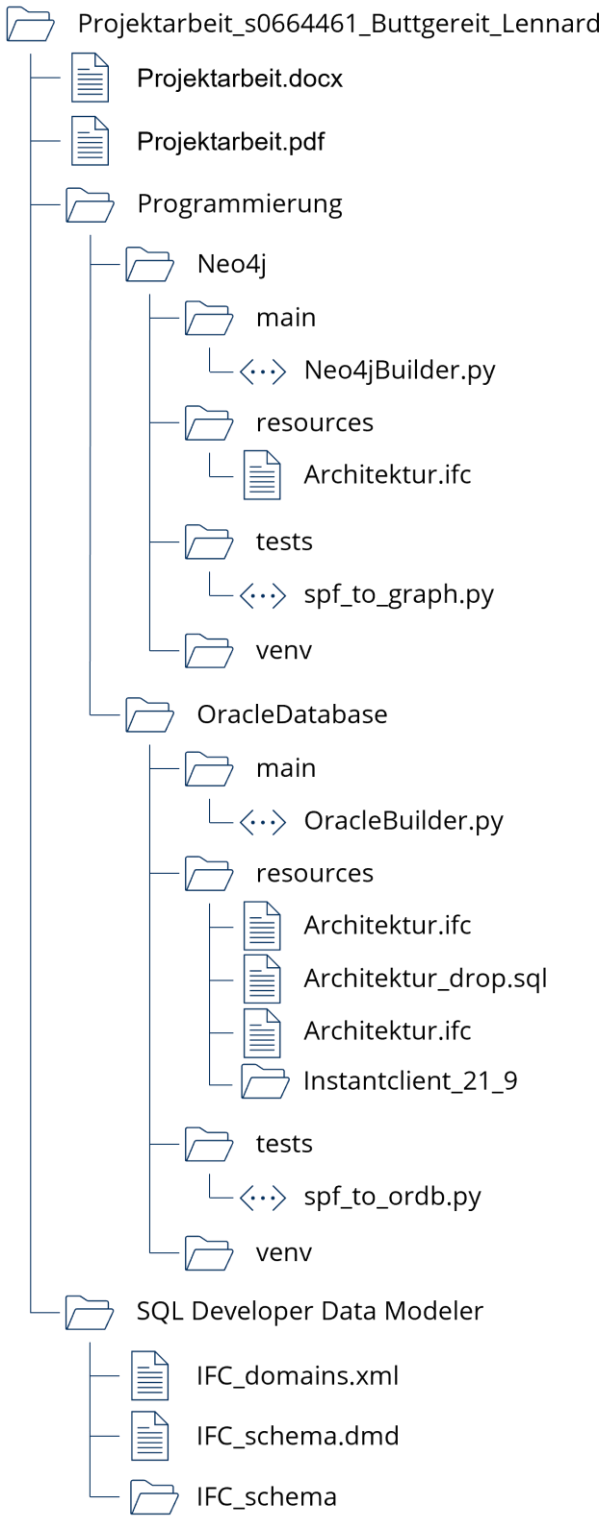
Zhu, J., Chong, H.-Y., Zhao, H., Wu, J., Tan, Y. & Xu, H. (2022). The Application of Graph in BIM/GIS Integration. *Buildings*, *12*(12), 2162. https://doi.org/10.3390/buildings12122162

VII ANLAGENVERZEICHNIS

Anlage 1 Digitales Anlagenverzeichnis i

Anlage 2 Beschreibung der digitalen Anlagen ii

ANLAGE 1 DIGITALES ANLAGENVERZEICHNIS



ANLAGE 2 BESCHREIBUNG DER DIGITALEN ANLAGEN

Projektarbeit_s0664461_Buttgereit_Lennard

- **Projektarbeit.docx**
  Der schriftliche Teil der Projektarbeit als Word-Dokument
- **Projektarbeit.pdf**
  Der schriftliche Teil der Projektarbeit als PDF-Dokument
- **Programmierung** (Ordner)
  Ordner für alle geschriebenen Programmteile
  - **Neo4j** (Ordner)
    Python-Projekt für die Transformation zu einem LPG in Neo4j
    - **main** (Ordner)
      - **Neo4jBuilder.py**
        Enthält alle benötigten Methoden zur Transformation (siehe Datei für Auflistung und Beschreibung der Methoden)
    - **resources** (Ordner)
      - **Architektur.ifc**
        Das zu transformierende BIM-Modell als IFC-SPF
    - **tests** (Ordner)
      - **spf_to_graph.py**
        Ausführbares Programm zur Transformation des IFC-SPF (siehe Datei für Beschreibung des Programmaufbaus)
    - **venv** (Ordner)
      Virtuelle Umgebung, die alle Fremd-Programmteile enthält
  - **OracleDatabase** (Ordner)
    Python-Projekt für die Transformation zu einem objekt-relationalen Datenbankschema in Oracle Database
    - **main** (Ordner)
      - **OracleBuilder.py**
        Enthält alle benötigten Methoden zur Transformation (siehe Datei für Auflistung und Beschreibung der Methoden)
    - **resources** (Ordner)
      - **Architektur.ifc**
        Das zu transformierende BIM-Modell als IFC-SPF
      - **Architektur_drop.sql**
        SQL-Befehle, die ein vorhandenes Schema bereinigen
      - **Architektur_schema.sql**
        SQL-Befehle, die das (leere) Schema erzeugen
      - **instantclient_21_9** (Ordner)
        Oracle Instant Client für erweiterte Netzwerkfunktionen

        - **tests** (Ordner)
            - **spf_to_ordb.py**
              Ausführbares Programm zur Transformation des IFC-SPF (siehe Datei für Beschreibung des Programmaufbaus)
        - **venv** (Ordner)
          Virtuelle Umgebung, die alle Fremd-Programmteile enthält
- **SQL Developer Data Modeler** (Ordner)
  Projektdaten zur Erzeugung des Datenbankschemas mit dem Oracle SQL Develoepr Data Modeler
    - **IFC_domains.xml**
      Importierbare Domains (Datentypspezifikationen), die die definierten Typen und Enumerationen des IFC-Metamodells abbilden
    - **IFC_schema.dmd**
      Mit dem Data Modeler ausführbare Datei zum Öffnen des Projektes
    - **IFC_schema** (Ordner)
      Projektdaten in Software-interner Struktur