Lehrstuhl für Computergestützte Modellierung und Simulation
Ingenieurfakultät für Bau Geo Umwelt
Technische Universität München

Konzeption einer raum-zeitlichen Anfragesprache für die Analyse und Prüfung von 4D-Gebäudeinformationsmodellen

Simon Fabian Daum

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. rer. nat. Ernst Rank
Prüfer der Dissertation:	1. Prof. Dr.-Ing. André Borrmann
	2. Prof. Dr. rer. nat. Thomas H. Kolbe

Die Dissertation wurde am 20.09.2017 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 23.01.2018 angenommen.

Zusammenfassung

Building Information Modeling (BIM) stellt einen computergestützten und modellbasierten Ansatz zur Interaktion mit Gebäuden über deren gesamten Lebenszyklus dar. Derzeit befindet sich die Bauindustrie in einer Übergangsphase zwischen der Anwendung traditioneller und neuartiger BIM-basierter Methoden. Trotz der inzwischen ausgereiften Datenmodelle für die Beschreibung von Bauwerken ergeben sich beim Einsatz von BIM technologische Hürden. So entstehen in einer computergestützten Planung ausgedehnte Datenbestände, deren Qualität über den Projekterfolg mitentscheidet. Gleichzeitig erweisen sich die manuelle Analyse und Überprüfung derartiger Datenbestände als zeitintensiv und fehleranfällig. Im Speziellen fehlt ein umfassendes Konzept, um dieses manuelle Vorgehen abzulösen und stattdessen robuste computergestützte Methoden einzusetzen.

Im Bereich der Geoinformatik kommen hingegen ausgereifte Analysetechniken zum Einsatz. Aus diesem Grund werden im Rahmen dieser Arbeit die Datenmodellierung und -verarbeitung sowohl in der Bauinformatik als auch in der Geoinformatik betrachtet. Anschließend wird untersucht, ob sich bestehende Anfragesprachen für die Analyse von BIM-Daten eignen. Auf Grund der erkannten Defizite werden als Erstes die benötigten domänenspezifischen Operatoren entwickelt. Anschließend erfolgt die Konzeption einer übergeordneten, sprachbasierten Analysemethode für Gebäudemodelle. In dieser können semantische, zeitliche und räumliche Aspekte kombiniert ausgewertet werden.

Um Fachanwendern die entwickelte Funktionalität in geeigneter Form zugänglich zu machen, beinhaltet das Konzept eine eigene domänenspezifische Anfragesprache. Das ebenfalls beschriebene Laufzeitsystem der Sprache stellt die Implementierungen der entwickelten Operatoren und eine Infrastruktur zur effizienten Geometrieverarbeitung bereit. Dafür kommen unter anderem Konzepte aus der algorithmischen Geometrie und der Geoinformatik zum Einsatz. Trotz der erweiterten Funktionalität beruhen die sprachbasierten Analysen auf einem verständlichen wiederkehrenden Muster. Dadurch wird der Einsatz der Sprache für Nutzer ohne tiefgreifende Informatikkenntnisse erleichtert. Neben einer textuellen Notation wird außerdem eine visuelle Sprachschnittstelle spezifiziert. So können Fachanwender eine formale und flexible Methode für die Analyse von Gebäudemodellen anwenden, ohne umfangreichen Programmcode schreiben zu müssen.

Durch die erweiterte BIM-spezifische Funktionalität und die Ausrichtung auf Fachanwender hebt sich das vorgestellte System von bereits bestehenden Ansätzen ab. Es erhöht nochmals den Mehrwert einer BIM-basierten Planung und erleichtert die Einführung dieser computergestützten Planungsmethode. Die prototypische Implementierung des Analysesystems wurde als Open Source Applikation Forschern und Anwendern aus der Industrie zugänglich gemacht.

Abstract

Building Information Modeling (BIM) represents a computer-aided and model-based approach for the interaction with buildings over their complete lifecycle. Currently, the construction industry is going through a phase between using traditional and novel BIM-based methods. Despite sophisticated data models for the description of buildings, technological hurdles may arise as BIM is applied. For example, in a computer-assisted planning, extensive datasets are created which quality contributes to the success of the project. At the same time, manual inspection and analysis of such datasets prove to be time-intensive and error-prone. In particular, a comprehensive concept is missing, which could replace this manual reviewing with robust computer-based methods.

In the field of geo-informatics, sophisticated techniques for data analysis are used instead. For this reason, data modeling and processing in construction- and in geo-informatics are considered. Thereafter, available query languages are examined whether they are suitable for the analysis of BIM data. Due to the identified deficits, required domain specific operators are developed. Subsequently, an overlying, language-based approach for analyzing building models is conceived. It enables to inspect semantic, temporal and spatial aspects of a model in combination.

To make the developed functionality accessible for the user in a suitable form, the system facilitates it own domain-specific query language. The presented runtime system of the language offers the implementation of the developed operators and incorporates an infrastructure for efficient geometric processing. For this, concepts taken from computational geometry and from geo-informatics are applied. Despite the extended functionality of the language, the definition of queries depends on a comprehensible, recurring pattern. This simplifies the application of the language for users without deep computational knowledge. To take this further, a visual notation is integrated in the language besides its textual interface. This allows professionals to use a formal and flexible method for the examination of building models, without writing extensive program code.

By the extended BIM-specific functionality and the orientation to domain experts, the presented system differs from existing concepts of analysis. It increases the value of the BIM-based planning and eases the introduction of this computational approach. To be used by researchers and professionals from the industry, the prototypical implementation of the analysis system was published as open source software.

Vorwort

Diese Arbeit entstand zwischen 2012 bis 2017 während meiner Tätigkeit am Lehrstuhl für Computergestützte Modellierung und Simulation der Technischen Universität München. Hier konnte ich unter exzellenten Bedingungen im Bereich der Bauinformatik forschen und neue Ansätze für das Building Information Modeling erarbeiten. Für diese Möglichkeit und für seine Unterstützung will ich mich daher sehr herzlich bei Professor André Borrmann bedanken. Ohne den stetigen Austausch mit ihm, seine wertvollen Anregungen sowie kritischen Hinweisen hätte ich diese Arbeit nicht in der jetzt vorliegenden Form verwirklichen können. Herrn Professor Thomas Kolbe möchte für die Übernahme des Zweitgutachtens, sein Interesse an meiner Arbeit und für die gemeinsame Forschung danken. Für ihre Unterstützung bedanke ich mich sehr herzlich bei Professor Ernst Rank und Professor Rafael Sacks sowie bei allen Kollegen an den Lehrstühlen CMS und CIE. Außerdem hat mir Joachim Schüller durch das Korrekturlesen der Arbeit sehr geholfen. Mein außerordentlicher Dank gilt meinen Eltern, meiner Partnerin Angela und meinen Freunden.

Formatierung

Hervorhebungen sind **fett** gedruckt. Fachwörter und Eigennamen werden bei erster Verwendung *kursiv* wiedergegeben. Dies gilt auch, wenn diese in einem neuen Kontext erscheinen. Für Klassenbezeichner wird die Schriftart *Latin Modern Sans Serif* verwendet. *Attributbezeichner* werden außerdem kursiv dargestellt. Die *Typwriter* Schriftart wird genutzt, wenn Teile des Programmcodes, beispielsweise `Variablennamen`, im Fließtext referenziert werden.

> Abschnitte mit zentralem Inhalten, wie Schlussfolgerungen und Fazits, werden grau hinterlegt dargestellt.

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D	Zweidimensional
3D	Dreidimensional
4D	Vierdimensional
4-IM	4-Intersection Model
9-IM	9-Intersection Model
AABB	Axis Aligned Bounding Box
ALB	Automatisiertes Liegenschaftsbuch
ALK	Automatisierte Liegenschaftskarte
ALKIS	Amtliches Liegenschaftskatasterinformationssystem
AST	Abstract Syntax Tree
BIM	Building Information Modeling
BIMQL	Building Information Model Query Language
BRep	Boundary Representation
BERA	Building Environment Rule and Analysis
BPMN	Business Process Modelling Notation
CAD	Computer Aided Design
CAFM	Computer Aided Facility Management
CG	Computational Geometry
CityGML	City Geography Markup Language
CSG	Constructive Solid Geometry
DBMS	Database Management System
DE9-IM	Dimension Extended 9-Intersection Model
DXF	Drawing Exchange Format
DL	Deskriptive Logik
EMF	Eclipse Modelling Framework
EBNF	Extended Backus-Naur Form
FLWOR	For, Let, Where, Order by, Return
GIS	Geoinformationssystem
GML	Geography Markup Language
HTML	HyperText Markup Language
ISO	International Organization for Standardization

IDM	Information Delivery Manual
IGES	Initial Graphics Exchange Specification
IFC	Industry Foundation Classes
IRI	Internationalized Ressource Identifier
LoD	Level of Detaill
LINQ	Language Integrated Query
JSON	JavaScript Object Notation
MOF	Meta Object Facility
MVD	Model View Definition
NURBS	Non-Uniform Rational B-Spline
OBB	Oriented Bounding Box
OGC	Open Geospatial Consortium
ODMG	Object Database Management Group
OQL	Object Query Language
OO	Objektorientierung/Objektorientiert
OWL	Web Ontology Language
P21	Part 21
Pixel	Picture Element
PMQL	Partial Model Query Language
PSD	Property Set Definition
QL4BIM	Query Language for 4D Building Information Models
^{t}QL4BIM	Textual Notation of QL4BIM
^{v}QL4BIM	Visual Notation of QL4BIM
QES	Query Expression Syntax
RDF	Resource Description Framework
RTS	Runtime System
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SRS	Spatial Reference System
SDAI	Standard Data Access Interface
STEP	Standard for the Exchange of Product model data
SQL	Structured Query Language
UDT	User Defined Type
UML	Unified Modeling Language
UoD	Universe of Discource
VarArgs	Variable Arguments
Voxel	Volume Element
W3C	World Wide Web Consortium
OWL	Web Ontology Language
XML	Extensible Markup Language

XSD	XML Schema Definition Language
XDM	XQuery and XPath Datamodel

Kapitel 1

Einführung

Der Entwurf und die Planung von Gebäuden ist heutzutage ohne computergestützte Methoden nicht mehr denkbar. Die bisherige Arbeitsweise beruht jedoch primär auf der Erstellung und dem Austausch von zweidimensionalen Zeichnungen, was eine Reihe von Einschränkungen hinsichtlich der digitalen Weiterverarbeitung von Informationen mit sich bringt. *Building Information Modeling (BIM)* setzt hingegen auf eine durchgängige Nutzung digitaler Modelle, in denen semantische und räumliche Daten kombiniert vorliegen. Momentan durchläuft das Bauwesen mit der Einführung von BIM einen tiefgreifenden technologischen Wandel. Um das Potential von Gebäudemodellen ausschöpfen zu können, bedarf es nach der hier vertretenen These zusätzliche Methoden zur Analyse, Verifikation und Weiterverarbeitung. Diese müssen auf die angewendete Datenmodellierung und die domänenspezifischen Gegebenheiten wie Datenumfang und -art angepasst sein. Die vorliegende Arbeit untersucht die aktuellen Modellierungskonzepte im BIM und im angrenzenden Fachgebiet Geoinformatik. Außerdem werden Methoden der Datenanalyse im Kontext unterschiedlicher Datenbanktechnologien diskutiert. So können relevante Ansätze aus der Geoinformatik und der allgemeinen Informatik identifiziert werden. Darauf aufbauend wurde ein umfassendes Konzept zur computergestützten Analyse, Verifikation und Weiterverarbeitung von BIM-Daten entwickelt. Dieses basiert auf einer domänenspezifischen Anfragesprache für Gebäudemodelle und einem entsprechenden Laufzeitsystem. Die entwickelten Konzepte unterstützen die Erstellung hochqualitativer Modelle und erleichtern die Einführung einer BIM-basierten Arbeitsweise.

1.1 Motivation und Thesen

Ein *Gebäudeinformationsmodell* (engl. *building information model*), in dieser Arbeit verkürzt als *Gebäudemodell* bezeichnet, stellt eine digitale, computerinterpretierbare Repräsentation eines bereits existierenden oder geplanten Bauwerks dar. Mit *Building Information Modeling (BIM)* wird eine spezielle, computergestützte Arbeitsweise während der Planung, Bau-

ausführung und Bewirtschaftung eines Gebäudes bezeichnet (Eastman et al., 2008). BIM propagiert dabei die Vorhaltung des Bauwerks als virtuelles Modell, das je nach Projektstatus erstellt, verfeinert und aktualisiert wird. Durch die eingesetzten Modellierungstechniken kann die *Semantik* von repräsentierten Entitäten vorgehalten und Beziehungen zwischen Entitäten abgebildet werden. Dreidimensionale Geometriedaten beschreiben zusätzlich die räumliche Lage modellierter Bauteile. Neben der Semantik von Entitäten und ihren geometrischen Repräsentationen spielen im Bauprozess auch geplante Konstruktions- und Einbauzeiten eine wichtige Rolle. Integriert man diese zeitlichen Daten, erhält man ein 4D-Gebäudemodell (McKinney & Fischer, 1998). Dreidimensionale und vierdimensionale Gebäudemodelle stellen die zu verarbeitende und zu analysierende Datenbasis in dieser Arbeit dar.

In den letzten Jahren wurde die Datenmodellierung in der Bauinformatik zunehmend ausgereifter. Dies ist unter anderem am offenen Transferformat *Industry Foundation Classes (IFC)* zu erkennen, das seit Mitte 2014 in Version 4 vorliegt und eine detaillierte Gebäudemodellierung erlaubt (Liebich, 2013). Die derzeitigen BIM-Konzepte reichen in der Praxis jedoch für eine wirtschaftliche Bearbeitung komplexer Planungsaufgaben nicht aus. Insbesondere fehlt ein herstellerneutraler Ansatz, um flexibel und effizient die relevanten Informationen aus einer BIM-basierten Datenbasis extrahieren zu können. Ohne diese Funktionalität müssen Selektionen, Analysen und Überprüfungen wiederholt manuell ausgeführt werden, obwohl eine formale Beschreibung der benötigten Verarbeitung möglich wäre. Beispielsweise erfordert bereits die Auswahl aller Instanzen eines Bauelementetyps manuelle Arbeitsschritte ohne die Möglichkeit auf Formalisierung. Des Weiteren werden komplexere Selektionsregeln, wie die Auswertung einer *Typhierarchie* und *räumliche Analysen* in den derzeit eingesetzten BIM-Werkzeugen nicht unterstützt.

Auch die zu anderen Industriezweigen abweichenden Randbedingungen in der Bauindustrie erfordern spezielle Analyse- und Weiterverarbeitungsmethoden. Als eine Besonderheit ist die Vielzahl an interagierenden Unternehmen und Gewerken innerhalb eines Bauprojekts zu nennen. Daher erstellt in einer modellbasierten Arbeitsweise eine heterogene Gruppe an Bearbeitern umfangreiche Gebäudemodelle. Qualitativ mangelhafte Teilarbeiten können innerhalb dieser verteilten Arbeitsweise unentdeckt bleiben, zumal ein hierfür benötigtes umfassendes Konzept für eine computergestützte, qualitative Analyse und Verifikation von Gebäudemodellen fehlt. In der Bauphase führt jedoch die Vernachlässigung der Modellqualität zu Verzögerungen, Mehrkosten und im Extremfall zum Scheitern des Projekts.

Die effiziente Informationsextraktion aus einer umfangreichen Datenbasis ist in der Informatik ein essentielles Forschungsthema. Hierzu existieren im Bereich der *Datenbanken* und ihrer *Anfragesprachen* bereits vielfältige und bewährte Konzepte. Die Analyse und Verifikation von Gebäudemodellen auf Basis der entsprechenden Standardtechnologien erweist sich jedoch als problematisch. So gestaltet sich die Definition von Anfragen kompliziert und für Fachanwender wie Architekten und Bauingenieure praktisch nicht handhabbar. Außerdem können lediglich

Teilaspekte eines Gebäudemodells untersucht werden, da benötigte Operatoren speziell für räumliche Untersuchungen fehlen.

In räumlichen Analysen werden anstelle der attributiven Daten die geometrischen Repräsentationen von Bauteilen verarbeitet. Die Kollisionskontrolle stellt die einzige etablierte Funktionalität in diesem Bereich dar. Hierbei wird überprüft, ob sich Geometrierepräsentationen innerhalb eines Modells durchdringen. Eine feingranulare Untersuchung unterschiedlicher räumlicher Beziehungen zwischen Bauteilen wird derzeit in BIM-Werkzeugen nicht unterstützt.

1.2 Lösungsansatz

Auf Grund der beschriebenen, eingeschränkten Analyse- und Validierungsunterstützung in der BIM-basierten Planung ist bereits bei Projekten mittlerer Größe eine durchgängige Qualitätsbeurteilung nur mit hohem personellem Aufwand zu realisieren. Diese arbeitsintensive, manuelle Modellüberprüfung wird deswegen häufig auf Grund betriebswirtschaftlicher Gesichtspunkten minimiert und ist zudem sehr fehleranfällig.

Im Rahmen dieser Arbeit wird eine domänenspezifische Anfragesprache und ein passendes Laufzeitsystem konzipiert. Dadurch wird angestrebt, das beschriebene technologische Defizit zu überwinden und eine umfassende und effiziente Analyse und Verifikation von 3D- und 4D-Modellen zu ermöglichen. Dies beinhaltet die kombinierte Untersuchung semantischer, räumlicher und zeitlicher Aspekte eines Gebäudemodells.

Im Bereich der Geoinformatik wurden bereits effiziente Methoden für die Analyse von zweidimensionalen Modellen mit Raumbezug entwickelt. In dieser Arbeit werden Konzepte vorgestellt, um derartige Analysen auf die Gegebenheiten des Building Information Modelings anzupassen. Dies mündet in der Entwicklung effizienter räumlicher Operatoren zur Auswertung topologischer, direktionaler und metrischer Beziehungen zwischen dreidimensionalen Bauteilrepräsentationen. Zur Bereitstellung der Analysefunktionalität werden die entworfenen Operatoren in die Anfragesprache integriert.

Im Gegensatz zu einer Reihe anderer Anfragesprachen folgt die hier vorgestellte Entwicklung dem Paradigma der imperativen Programmierung. Grammatikalisch basiert die Sprache auf einer reduzierten Anzahl an Sprachregeln, stellt jedoch gleichzeitig eine formale und generelle Methodik für BIM-Analysen dar. Der überwiegende Teil an Anfragesprachen lässt sich zudem ausschließlich durch eine textuelle Notation anwenden. Dies kann für Fachanwender, die an graphische Benutzerschnittstellen gewöhnt sind, ein Hindernis darstellen. Daher unterstützt die Anfragesprache neben einer textuellen auch eine visuelle Notation. In dieser werden Anfragen durch graphische Elemente und deren gegenseitigen Verbindungen repräsentiert.

Wie bereits erwähnt, wird neben der Anfragesprache auch ein entsprechendes Laufzeitsystem vorgestellt. Dieses ist speziell für die effiziente Analyse umfangreicher Modelle ausgelegt. Als Eingangsquelle unterstützt das Analysesystem detaillierte Gebäudemodelle. Um Planungsdaten auch in einem erweiterten geographischen Kontext analysieren zu können, besteht zudem die Möglichkeit, Stadtmodelle zu verarbeiten und Anfragen über die Modellgrenzen hinweg auszuführen.

Somit können die zentralen Thesen dieser Arbeit formuliert werden:

- Auf Grund der Komplexität des abzubildenden Informationsraums ist der Einsatz einer formalen Anfragesprache für die Analyse und Verarbeitung von BIM-Daten sinnvoll.
- Trotz eines generellen Ansatzes zur BIM-Analyse kann die Anfragesprache so konzipiert werden, dass sie durch Fachanwender eingesetzt werden kann.
- Eine derartige Anfragesprache benötigt spezielle Operatoren, um semantische, relationale, räumliche und zeitliche Modellaspekte auswerten zu können.
- Auch bei der Verarbeitung detaillierter räumlicher Repräsentationen können Ausführungszeiten erreicht werden, die den praktischen Einsatz der Sprache bei umfangreichen Modellen ermöglicht.
- Durch die Entwicklung toleranzunterstützender Erweiterungen für räumliche Operatoren können die Einsatzmöglichkeiten topologischer und direktionaler Operatoren im ingenieurtechnischen Umfeld erweitert werden.

1.3 Aufbau der Arbeit

In Kapitel 2 wird auf Methoden der Geoinformatik und auf Geoinformationssysteme eingegangen. Insbesondere wird die räumliche und zeitliche Datenmodellierung besprochen. Anschließend wird auf das Transferformat *City Geography Markup Language (CityGML)* für die Repräsentation von Stadtmodellen eingegangen. Nach Betrachtung der in diesem Fachgebiet gängigen Analysefunktionen werden Datenstrukturen zur *räumlichen Indizierung* vorgestellt. Kapitel 3 untersucht die Konzepte des Building Information Modelings. Nach einer einführenden Betrachtung der Modellierungsstrategien im Bauwesen wird das Datenmodell *Industry Foundation Classes (IFC)* erörtert. Die in dieser Arbeit entwickelten Analysekonzepte nutzen das IFC-Modell als primäre Datenquelle. Abschließend wird die Terminplanung und die damit einhergehende zeitliche Modellierung im BIM betrachtet.

In Kapitel 4 wird untersucht, inwiefern gängige Datenbankmodelle und Anfragesprachen als umfassende BIM-Analysewerkzeuge genutzt werden können. In diesem Kapitel werden zusätzlich domänenspezifische Anfragesprachen, die aktueller Gegenstand der Forschung sind, besprochen. Auf Basis der identifizierten Defizite bisheriger Ansätze werden in Kapitel 5 fehlende Operatoren und Algorithmen für eine BIM-Anfragesprache vorgestellt. Diese bilden einen wichtigen Teil der domänenspezifischen *Query Language for 4D Building Information Models (QL4BIM)*. Die Sprache wurde im Rahmen dieser Arbeit entwickelt und wird in Kapitel 6 detailliert vorgestellt. Dabei wird unter anderem auf das Programmierkonzept, die Sprachgrammatik und das Typsystem eingegangen. Außerdem werden zwei Notationen für die Erstellung von Anfragen vorgestellt. Die benötigte Ablaufumgebung der Sprache wird im nachfolgenden Kapitel 7 diskutiert.

In Kapitel 8 wird die Einsatzeignung der Sprache untersucht. Die Arbeit schließt mit einem Überblick bezüglich des entwickelten Konzepts einer raum-zeitlichen Anfragesprache für die Prüfung und Analyse von 4D-Gebäudemodellen ab und erläutert mögliche nachfolgende Forschungsbereiche.

Kapitel 2

Geoinformatik und Geoinformationssysteme

Während Baumaßnahmen sind nicht nur detaillierte Daten der geplanten Bauwerke nötig. Auch geographische Daten des beeinflussten Bereichs müssen betrachtet werden. Zur rechnergestützten Vorhaltung ausgedehnter räumlicher Daten stellt die Geoinformatik vielfältige Methoden bereit. Dabei ergeben sich Überschneidungen zwischen den Methoden der Datenmodellierung und -verarbeitung im Bauwesen und in der Geoinformatik, da in beiden Disziplinen räumliche und zeitliche Phänomene hohe Bedeutung einnehmen. Gleichzeitig existieren in der Geoinformatik ausgereifte Konzepte zur Analyse und Verifikation der vorgehaltenen raum-zeitlichen Daten. Diese sind in der BIM-basierten Gebäudemodellierung nicht gleichermaßen vorhanden. Eine These dieser Arbeit besteht darin, dass eine derartige Funktionalität in angepasster Form auch im Building Information Modeling zur Produktivitäts- und Qualitätssteigerung eingesetzt werden kann.

In diesem Kapitel wird daher auf die räumliche und zeitliche Datenmodellierung in der Geoinformatik eingegangen. Des Weiteren wird das CityGML-Format für die Repräsentation von Stadtmodellen vorgestellt. Dieses kann neben den primär zu verarbeitenden BIM-Daten als zusätzliche Datenquelle innerhalb des entwickelten Analysesystems genutzt werden. Das Kapitel schließt mit der Betrachtung der in Geoinformationssystemen verfügbaren Analysefunktionen und mit der Besprechung ausgewählter räumlicher Indizierungstechniken ab. Beides fließt in die Konzeption der domänenspezifischen Anfragesprache und des entsprechenden Laufzeitsystems ein.

2.1 Grundlagen

Die Geoinformatik lässt sich als interdisziplinäres Fachgebiet verstehen, das eine Brückenfunktion zwischen Informatik, geographischen Informationstechnologien und raumbezogen arbeitenden Wissenschaften einnimmt (Lange, 2013, S. 1). Eine ausführliche Definition enthält Bill & Zehner (2001, S. 110):

> „Die Geoinformatik setzt sich mit dem Wesen und der Funktion der Geoinformation, mit ihrer Bereitstellung in Form von Geodaten und mit den darauf aufbauenden Anwendungen auseinander. Die dabei gewonnenen Erkenntnisse münden in die Geo-Informationssysteme. Allen Anwendungen der Geoinformatik gemeinsam ist der Raumbezug."

Raumbezogene Daten lassen sich folglich in einem *Geoinformationssystem (GIS)* abbilden und mit Methoden der Geoinformatik gezielt auswerten. Ein GIS kann als spezialisiertes Informationssystem betrachtet werden. Diese Systeme verwirklichen das EVAP-Modell, das Funktionen zur **E**rfassung, **V**erwaltung, **A**nalyse und **P**räsentation von Daten beinhaltet (Kappas, 2011).

In den letzten Jahren konnten Softwarehersteller und Anwender von zunehmend standardisierten Modellierungsmethoden im GIS-Bereich profitieren. Die zwei wichtigsten Normierungsgremien in der Geoinformatik sind das *Technical Committee TC211* der International Organization for Standardization (ISO) und das industrienahe *Open Geospatial Consortium (OGC)*. Innerhalb des folgenden Abschnitts wird unter anderem auf die *ISO-19125 (Simple Feature Access)*, die *IS0-19107 (Spatial Schema)* und das OGC-spezifizierte Datenmodell CityGML eingegangen. Abbildung 2.1 zeigt eine Übersicht über die hier vorgestellten Standards und gegenseitigen Beziehungen.

2.2 Datenmodellierung in Geoinformationssystemen

In einem GIS werden Objekte der Umwelt in ihrem gemeinsamen räumlichen Bezugsrahmen abgebildet und lassen sich über ihren inhärenten Raumbezug auswerten. Eine komplette computergestützte Abbildung der realen Welt ist jedoch nicht möglich. Daher bedient man sich einer Anforderungsanalyse für das zu erstellende System. Durch eine Reduktion auf den benötigten Ausschnitt der Realwelt, im Englischen als *Universe of Discource (UoD)* bezeichnet, können handhabbare Datenstrukturen realisiert werden (Shi et al., 2002, Kap. 13.5). Für eine detaillierte Beschreibung, der während dieser Modellbildung eingesetzten Abstraktionsschritte, sei auf (Partsch, 2010, Kap. 2.1.5) verwiesen.

Ein Geoinformationssystem nutzt in der Regel eine Datenbank als interne Komponente, um Geodaten und beschreibende Sachdaten persistent zu speichern (Güting, 1994, Kap. 5). Die

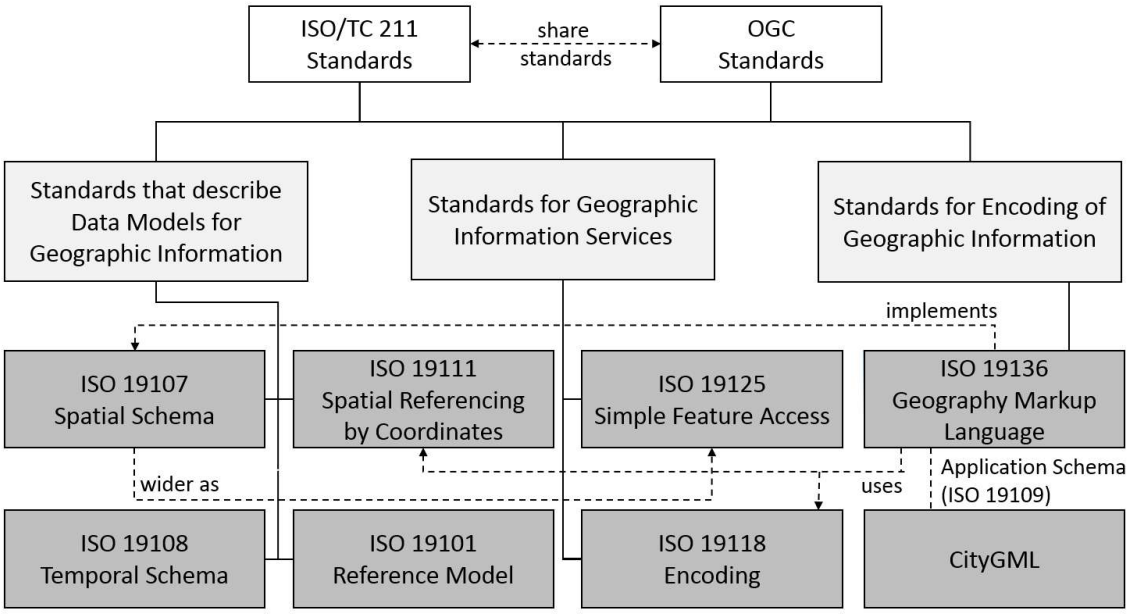


Abbildung 2.1: Die in diesem Kapitel behandelten ISO- und OGC-Standards und ihre gegenseitigen Beziehungen, nach (Tom & Roswell, 2009; OGC, 2012)

hier angewendete Datenmodellierung basiert daher auf Methoden, wie sie im Entwurf von Datenbankanwendungen und komplexen ingenieurtechnischen Systemen zum Einsatz kommen. Innerhalb der Planung eines Geoinformationssystem werden dementsprechend mehrere Modellierungsschritte auf unterschiedlichen Abstraktionsniveaus durchlaufen. Dabei entsteht nacheinander ein konzeptuelles, ein logisches und ein physisches Modell (Oppel, 2010, Kap. 1). Konzeptionelle Modelle können nach (Partsch, 2010, S. 37) folgendermaßen beschrieben werden:

> „Konzeptionelle Modelle beschreiben Systeme auf der Ebene menschlicher Konzeptdarstellung. Sie werden vor allem verwendet, um ein Verständnis für Zusammenhänge in großen, komplexen Systemen zu erhalten."

Ein konzeptionelles Modell wird in einer implementierungsunabhängigen Repräsentation vorgehalten. Dazu können beispielsweise die Modellierungssprache *Unified Modeling Language (UML)* und das *Entity-Relationsship-Modell* eingesetzt werden (OGC, 2011; Chen, 1976). In der darauffolgenden logischen Datenmodellierung wird auf Grund des ausgewählten Datenbanktyps eine weitere Verfeinerung des Modells vorgenommen. Der physische Entwurf ist von der konkret eingesetzten Datenbankimplementierung, Parametern des Betriebssystems und der genutzten Hardware abhängig. Hierbei wird beispielsweise die verwendete Indizierung und das *Clustering* zusammenhängender Daten spezifiziert (Elmasri et al., 2002, S. 589).

2.2.1 Räumliche Bezugssysteme

In einem Geoinformationssystem werden Objekte über ihren einheitlichen Raumbezug in einen gemeinsamen Kontext gebracht. Dazu ist ein eindeutig definiertes Raumbezugssystem, im Englischen als *Spatial Reference System (SRS)* bezeichnet, nötig (Lother, 2003, Kap. 4). Dieses beinhaltet eine mathematisch beschreibbare Referenzfläche auf der ein Koordinatensystem ausgebreitet wird. Zur Realisierung eines Raumbezugssystem muss neben dem Bezugsrahmen auch die Lage der Referenzfläche zum Erdkörper definiert werden. Dies geschieht über das *Geodätische Datum*, das die Position des Ursprungs, die Skalierung und die Orientierung des Koordinatensystems auf der Bezugsfläche über Parameter definiert.

Je nach Anwendungsfall werden in einem GIS unterschiedliche Raumbezugssysteme genutzt (Bill, 2010, Kap. 3.5). Dazu gehören unter anderem *geozentrische*, *ellipsoidische*, *geodätische* und *kartesische* Systeme. Je nach verwendetem System ergeben gleiche Operationen, wie beispielsweise eine Distanzmessung, abweichende Ergebnisse. Dies ist auf die unterschiedlichen mathematischen Raumdefinitionen zurückzuführen. In Abbildung 2.2 werden ein geodätisches und ein kartesisches Raumbezugssysteme gegenübergestellt.

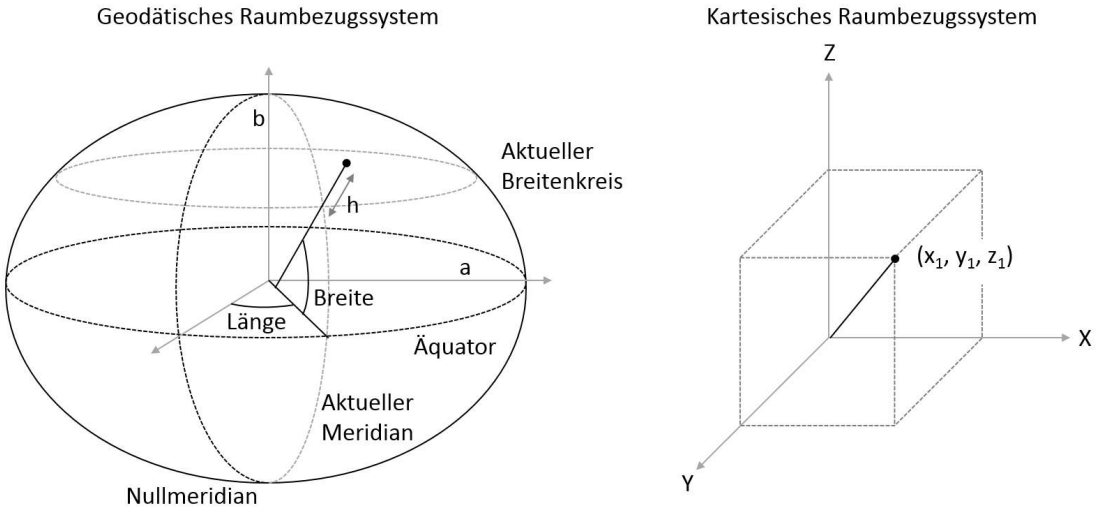


Abbildung 2.2: Gegenüberstellung eines geodätischen und eines kartesischen Systems, nach (Lange, 2013, S. 142) und (Bartelme, 2005, S. 202)

Auf Grund der geringen räumlichen Ausdehnung von Gebäudemodellen sind im Bauwesen lokale, kartesische Raumbezugssysteme der Standard. In diesen Systemen wird die Erdkrümmung nicht berücksichtigt. Räumliche Berechnungen sind jedoch effizient lösbar, da keine sphärische Trigonometrie angewendet werden muss. Wegen der Ausrichtung auf die im Bauwesen üblichen Datenbestände wird bei der Betrachtung

räumlicher Fragestellungen im Rahmen dieser Arbeit stets von einem lokalen, kartesischen Koordinatensystem ausgegangen.

Im Kontext der Raumbezugssysteme sei abschließend auf die *ISO-19111* hingewiesen, die als abstrakte Spezifikation ein konzeptionelles Schema zur Definition von 1- bis 3-dimensionalen räumlichen Koordinatensystemen liefert. Dabei ist zusätzlich die Einbindung einer zeitlichen Dimension möglich. Hierzu kann ein zusammengesetztes Koordinatensystem genutzt werden, in dem beispielsweise das zeitliche Schema aus der *ISO-19108* integriert wird (ISO, 2007b, 2002b). Die Modellierung zeitbehafteter Daten in Geoinformationssystemen wird im Abschnitt 2.2.4 dieses Kapitels behandelt.

2.2.2 Geometrische Datenmodellierung

Im Kontext der Modellbildung in Geoinformationssystemen ist der Begriff *Feature* zu erläutern. Die *ISO-19101* definiert ein Feature als die Abstraktion eines Phänomens, das in der Realwelt existiert (ISO, 2014, Kap. 4.11). Zur Vorhaltung der geometrischen Repräsentation eines Features haben sich in der Geoinformatik drei Modelle etabliert, die im Folgenden besprochen werden (Zimmermann, 2012, S. 20; Lange, 2013, S. 26).

Im *Vektormodell* werden Randpunkte durch Kanten verbunden und Flächen durch Kanten definiert. Dadurch ergibt sich eine randbasierte Repräsentation für jedes Feature. Wird dieses Konzept auf drei Dimensionen erweitert, können räumliche Körper durch eine Aggregation ihrer Randflächen dargestellt werden. Das *Rastermodell* basiert auf regelmäßig angeordneten Rasterzellen. Jede Zelle beinhaltet einen diskreten Wert und kann durch ihren Reihen- und Spaltenindex angesprochen werden. Im zweidimensionalen Anwendungsfall wird eine Zelle auch als *Picture Element (Pixel)* bezeichnet. Werden im Rastermodell drei Dimensionen abgebildet, stellt ein *Volume Element (Voxel)* das verwendete Primitiv dar. Typische Beispiele für Rasterdaten sind *Orthophotos*. Überlagert man ein Rastermodell mit einem Vektormodell, erhält man ein *hybrides Modell*, das aus Navigationsanwendungen bekannt ist. Hier werden Transportwege über das Vektormodell repräsentiert und die Umgebung als Rastermodell vorgehalten.

Neben der Klassifizierung nach Vektor-, Raster- oder Hybrid-Modell lässt sich die Datenmodellierung innerhalb eines Geoinformationssystems auch durch die Anzahl an vorgehaltenen geometrischen Dimensionen charakterisieren (Ott & Swiaczny, 2001, Kap. 2.5). Ein zweidimensionaler Ansatz basiert auf der Speicherung von Koordinatenpaaren für X- und Y-Werte ohne eine Höhenangabe. Dieser kann durch die Hinzunahme eines digitalen Geländemodells oder eines Attributs für Höhenwerte erweitert werden, wodurch sich eine *2.5D-Modellierung* ergibt. Eine dreidimensionale Modellierung liegt vor, wenn für alle Punkte Höhenwerte vor-

handen sind und Feature als Volumenkörper abgebildet werden können. Zur Speicherung von weiteren Daten wie Zeitinformationen und Messwerten hat sich die Verwendung eines zusätzlichen Parameters in jedem Koordinatentuple etabliert. Häufig wird dieser Parameter als T oder M bezeichnet (Herring, 2011a, S. 65).

Auf Grund der fachlichen Nähe zur Kartographie und der eingesetzten Datenerfassungsmethoden basiert die Mehrzahl an Geoinformationssystemen noch auf einem zweidimensionalen Geometriemodell. So wurde beispielsweise das amtliche Liegenschaftskataster in Deutschland bisher durch die 2D-basierten Anwendungen *Automatisiertes Liegenschaftsbuch (ALB)* und *Automatisierte Liegenschaftskarte (ALK)* realisiert. Dennoch ist die Tendenz zu einer attributiven Vorhaltung von Höheninformationen bzw. einer kompletten 3D-Modellierung zu erkennen. So erfolgt zur Zeit eine Umstellung des amtliche Liegenschaftskatasters auf das *Amtliches Liegenschaftskatasterinformationssystem (ALKIS)*, welches die Vorhaltung von 3D-Geometrie für Gebäude unterstützt (Gerschwitz, 2011). Zudem steigt die Verfügbarkeit von dreidimensionalen Stadtmodellen, die als CityGML-Instanzen vorliegen.

Die modellbasierte Planung im Bauwesen nutzt ihrerseits das dreidimensionale Vektormodell für die Geometrierepräsentation von Bauteilen und räumlichen Strukturen. Daher sind für diese Arbeit die Methoden der Geoinformatik für 3D-Vektormodelle ausschlaggebend und es wird nicht weiter auf Raster-, Hybrid-Modelle und zweidimensionale Vektormodelle eingegangen.

Für die Repräsentation von Vektorgeometrie in Geoinformationssystemen existieren mehrere OGC-Spezifikationen, die zusätzlich als ISO-Standards veröffentlicht wurden. OGC-Spezifikationen werden nach Abstraktionsgrad unterschieden. So stellt eine abstrakte Spezifikation einen konzeptionellen Entwurf dar, wohingegen Implementierungsspezifikationen konkretere technische Vorgaben beinhalten.

Die Simple Feature Access-Spezifikation (ISO-19125) definiert ein Objektmodell und Methoden für Vektorgeometrie. Eine topologische Modellierung wird nicht unterstützt (Herring, 2011b). Grundlage bilden Klassen für die Repräsentation von Punkten, Polygonen und Oberflächen in bis zu vierdimensionalen Koordinatensystemen. Die vierte Dimension kann hierbei über einen *Messurement*-Wert M repräsentiert werden, der beispielsweise die Stationierung auf einem linienhaften Bauteil aufnehmen kann. Simple Features sind des Weiteren dadurch charakterisiert, dass Krümmungen durch lineare Interpolation realisiert werden.

Als Implementierungsspezifikation realisiert die ISO-19125 teilweise das abstrakte, umfangreichere Spatial Schema (ISO 19107, 2003). Dieses beinhaltet neben den erwähnten Geometrietypen auch dreidimensionale Körper und erlaubt die Repräsentation gekrümmter Linien und Flächen. Durch *Composite*s können mehrere Geometrieobjekte dieselben Subelemente beinhalten und *Aggregate*s unterstützen die Gruppierung von geometrischen Instanzen. Au-

ferdem ist die Vorhaltung expliziter, topologischer Strukturen möglich. Auf die topologische Modellierung in Geoinformationssystemen wird im nachfolgenden Abschnitt eingegangen.

2.2.3 Topologische Datenmodellierung

Die mathematische Definition des Begriffs *Topologie* lautet nach Forster (2013, S. 9): Sei X eine Menge. Eine Menge T von Teilmengen von X heißt Topologie auf X, falls gilt:

- $\emptyset, X \in T$.
- Sind $U, V \in T$, so gilt auch $U \cap V \in T$.
- Ist I eine beliebige Indexmenge und $U_i \in T$ für alle $i \in I$, so folgt $\bigcup_{i \in I} U_i \in T$.

Aufbauend auf diesen Grundlagen wurden weitreichende mathematischen Konzepte entwickelt, um Eigenschaften räumlicher Objekte zu beschreiben, ohne eine explizite Metrik des Raumes nutzen zu müssen (Armstrong, 1983; Egenhofer & Herring, 1992; Clementini et al., 1993). Die topologischen Primitive sind *Knoten*, *Kanten*, *Maschen* und *Körper*. Abbildung 2.4 demonstriert *topologische Assoziationen* anhand der Modellierung von Flurstücken.

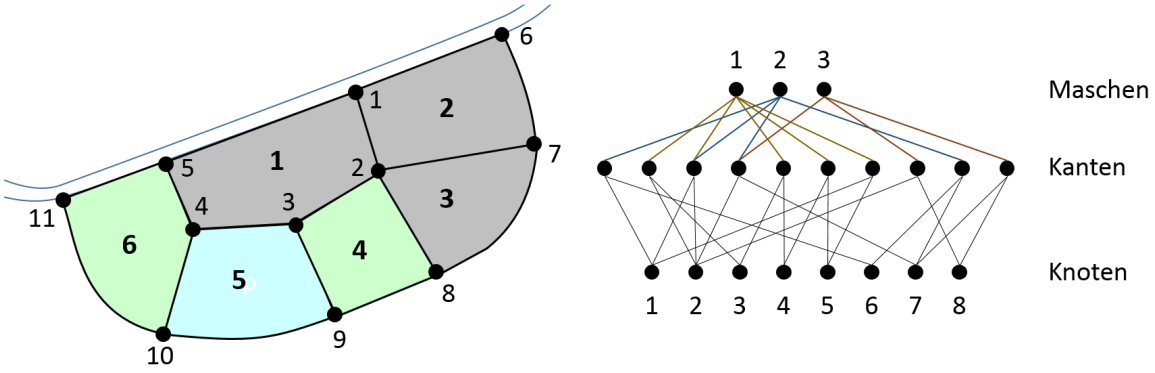


Abbildung 2.3: Topologische Assoziationen, nach (Lother, 2007, Kap. 5)

Innerhalb der Topologie wird zwischen Assoziationen und Relationen unterschieden (Lother, 2007, S. 4 - 11). Topologische Assoziationen treten **innerhalb** eines topologischen Primitivs auf und beschreiben das Zusammenspiel der beteiligten untergeordneten Primitiven.

Topologischen Relationen herrschen hingegen **zwischen** räumlichen Primitiven. Auf Grund ihrer Vielfalt werden sie im Standardfall durch Auswertung *topologischer Invarianten* dynamisch aus dem Datenbestand ermittelt. Zu den topologischen Invarianten in zweidimensionalen Räumen zählen *Geschlossenheit*, *Schnittpunkttreue*, *Trennung zwischen Innen und Außen* sowie die *Randpunkteigenschaft* (Egenhofer & Franzosa, 1991). Topologische Assoziationen

sind demnach explizit durch das verwendete topologische Datenmodell gegeben, wobei topologische Relationen dynamisch berechnet werden müssen. Auf topologische Relationen und ihrer formalen Analyse wird in Abschnitt 2.4.2 eingegangen.

Das Spatial Schema (ISO-19107) liefert neben einem geometrischen auch ein topologisches Modell. Für topologische Typen wird die Basisklasse TP_Object genutzt, von der Subklassen für topologische Primitive, wie Knoten, Kanten, Maschen und Körper ableiten. Diese liegen sowohl in einer ungerichteten als in einer gerichteten Variante vor. Abbildung 2.4 zeigt eine Kombination einer geometrischen und topologischen Modellierung innerhalb der ISO-19107.

In (Clemen, 2010, S. 15) werden als Vorteile einer topologischen Modellierung eine mögliche Performancesteigerung und die Überprüfbarkeit der Konsistenz topologischer Instanzen genannt. Die topologischen Klassen des Spatial Schemas können somit eine geometrische Modellierung unterstützen. Beispielsweise ist die Nachbarschaft zwischen den zwei Flächen in Abbildung 2.4 ohne die Verarbeitung geometrischer Daten mit Hilfe *kombinatorische Algorithmen* erkennbar.

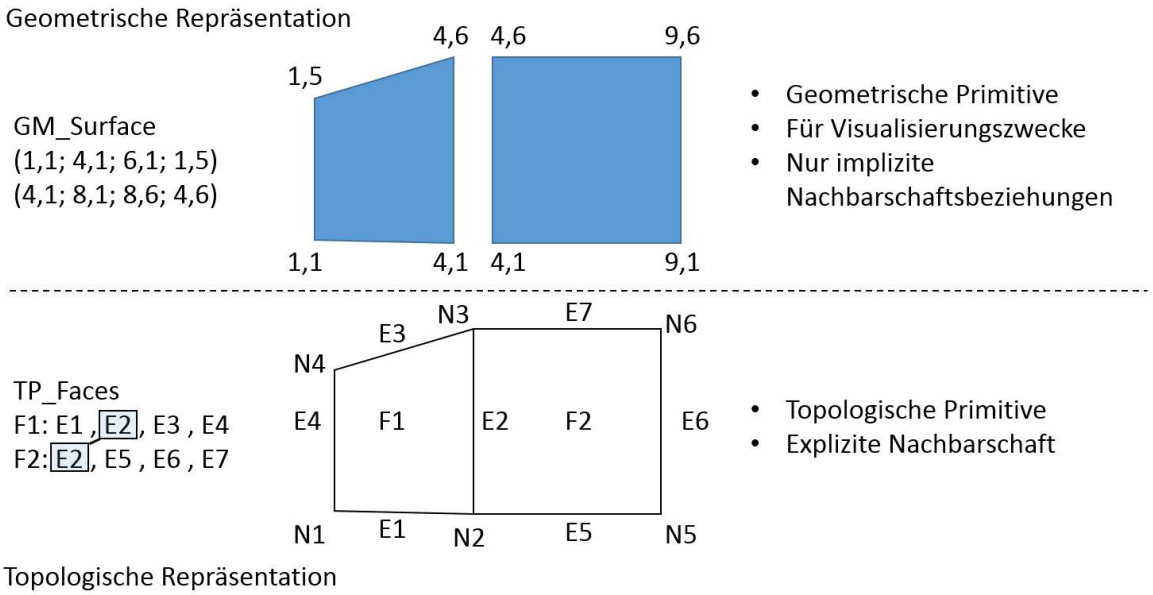


Abbildung 2.4: Geometrische und topologische Modellierung im Spatial Schema, nach (Reinhardt, 2011, S. 32)

Topologische Relationen zwischen Objekten können einen vorteilhaften Zugang zu Gebäudemodellen liefern. Die Vorhaltung aller derartiger Beziehungen als explizite Relationen erscheint jedoch nicht praktikabel und würde einer kompakten Modellrepräsentation entgegenwirken. Daher wird ein Konzept zur effizienten Auswertung topologischer Relationen auf Basis der vorhandenen geometrischen Daten benötigt. Ein entsprechender Lösungsvorschlag wird in Kapitel 5.2 diskutiert.

2.2.4 Zeitliche Datenmodellierung

Standardmäßig wird ein Geoinformationssystem genutzt, um ein einziges Abbild der Wirklichkeit vorzuhalten. Die zugrundeliegende Datenbasis spiegelt dabei den Zustand zum Zeitpunkt der jeweiligen Datenaufnahme wider. Im Zuge neuer Erhebungen kommt die Frage auf, wie mit veralteten Daten umgegangen wird. Besonders bei Geodaten, deren Beschaffung mit hohen Personalaufwand und Kosten verbunden ist, erscheint eine Überschreibung alter Bestände problematisch. Des Weiteren erfordert die Erschließung einer Vielzahl an räumlichen Phänomenen die Betrachtung zeitlicher Aspekte.

Aus diesem Grund ist die Integration der zeitlichen Modellierung ein wichtiges Forschungsfeld in der Geoinformatik (Copeland, 1980, Kap. 2.1). Dabei werden zwei zentrale Fragestellungen unterschieden (Peuquet, 1999, Kap. 2.1) :

1. *World State*: Wie stellt sich der gesamte Datenbestand bzw. eine Teilmenge einzelner Feature zu einem bestimmten Zeitpunkt dar?
2. *Change*: Welche Veränderungen treten im Datenbestand bzw. bei einzelnen Featuren zwischen zwei oder mehreren Zeitpunkten auf?

Wie die räumliche Modellierung benötigt die zeitliche Modellierung spezielle Datentypen. Snodgrass (1986, S. 2 - 3) führt hierfür Instant, Event, Time Period, Time Interval und Temporal Element ein. Tabelle 2.1 verdeutlicht die Semantik dieser Typen. Der Autor macht darauf aufmerksam, dass die Unterscheidung zwischen Perioden und Intervallen in der Literatur nicht eindeutig gehandhabt wird. In dieser Arbeit werden beide Varianten als Zeitintervalle bezeichnet.

Zeitlicher Typ	Beschreibung
Instant	Punkt auf der Zeitachse; Zeitpunkt
Event	Ein Ereignis, das bei einem bestimmten Zeitpunkt auftritt
Time Period	Eine Zeitspanne zwischen zwei Zeitpunkten
Time Interval	Eine Zeitspanne, die ausschließlich über eine Dauer definiert wird
Temporal Element	Eine Aggregation mehrerer Time Periods

Tabelle 2.1: Grundlegende zeitliche Typen, nach (Snodgrass, 1986)

Neben diesen grundlegenden Datentypen kommen in einer zeitlichen Datenmodellierung außerdem zwei unterschiedliche Zeitbezugssysteme zum Einsatz (Dyreson et al., 1994, Kap. 3). Die *Valid Time* wird als die Zeit definiert in der ein Fakt, der innerhalb des Systems abgelegt ist, Gültigkeit aufweist. Daten dieses Zeittyps müssen vom Benutzer bereitgestellt werden. Als *Transaction Time* wird die Zeit bezeichnet, in der ein Fakt innerhalb des Systems verfügbar ist. Diese Zeitspanne wird bestimmt durch die Zeitpunkte an dem der Fakt in das System

eingefügt bzw. gelöscht wird. Die Löschung muss hierbei nicht physikalisch erfolgen, sondern kann auch symbolisch angezeigt werden. Daten dieses Zeittyps können durch das Informationssystem aufgezeichnet werden und spiegeln die Folge von ausgeführten Transaktionen wider.

In der Forschung wurde eine Vielzahl konzeptioneller Modelle für raum-zeitliche Informationssysteme entwickelt und diskutiert. Die drei wichtigsten werden im folgenden besprochen (Langran & Chrisman, 1988, S. 8-12).

Das *Snapshot Model* kann als einfachstes der raum-zeitlichen Modelle angesehen werden. Hierbei werden Momentaufnahmen des Datenbestandes mit einer Zeitangabe versehen und chronologisch in *Ebenen* (engl. *layers*) des Systems abgelegt (Abbildung 2.5). Das Datenmodell leidet an der hohen Redundanz der gespeicherten Daten und dem Fehlen von expliziten Informationen über die aufgetretenen Änderungen zwischen unterschiedlichen Layern.

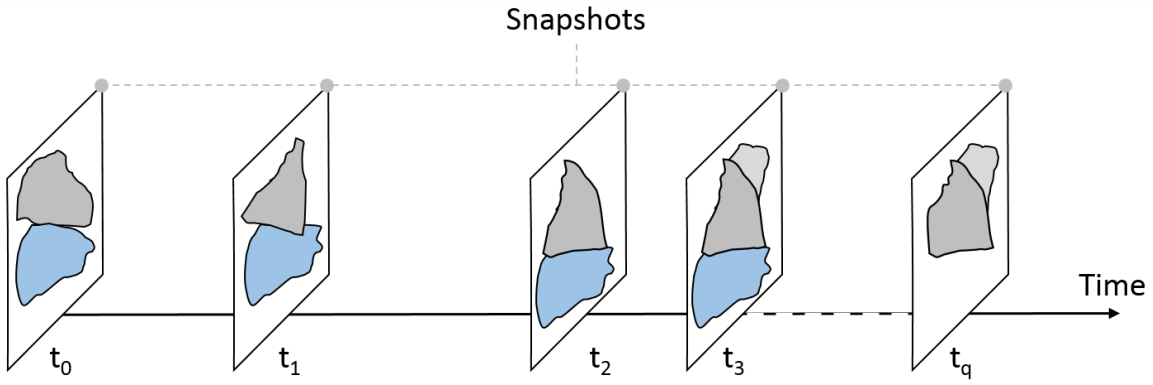


Abbildung 2.5: Darstellung der chronologisch angeordneten Layer des Snapshot Models, nach (Langran & Chrisman, 1988, S. 8)

Das *Update Model* basiert auf der kompletten Speicherung eines Datenbestand bezüglich des initialen Zeitpunkts t_0. Für die Datenaufnahme zum späteren Zeitpunkt t_i werden ausschließlich Änderungen vorgehalten. Dabei kann die Änderung entweder zum Datenbestand des vorherigen Zeitpunkts t_{i-1} oder zum initialen Bestand von t_0 berechnet werden. Wird hauptsächlich der aktuellste Datenbestand angefragt, kann es zur Vermeidung aufwendiger Berechnungen sinnvoll sein, stets diesen explizit vorzuhalten und die entsprechende Änderungshistorie für frühere Zeitpunkte zu speichern. Die Vorteile gegenüber dem Snapshot Model sind die Vermeidung redundanter Daten und eine explizite Vorhaltung der Veränderungen zwischen den Aufnahmezeitpunkten.

Das *Space Time Composite Model* besitzt Ähnlichkeiten zum Update Model, da auch hier die Änderungen zwischen Zeitpunkten gespeichert werden. Anders als im vorherigen Ansatz wird jede Änderung separat betrachtet und als eigenständiges Objekt modelliert. Dabei entstehen neue Objekte, die ihre Attributdefinitionen von ihren zugrundeliegenden Elternobjekten erben. Die eigentlichen Attributwerte werden für jeden Erhebungszeitpunkt gespeichert und sind über einer Historie abrufbar.

Neben diesen drei früh entwickelten und in der Forschung häufig genannten raum-zeitlichen Modellierungen existiert noch eine Vielzahl weiterer Ansätze. So werden von Pelekis et al. (2004, Kap. 3) insgesamt zehn Konzepte für ein zeitbehaftetes GIS identifiziert. Aus Sicht einer BIM-basierten Datenmodellierung erweisen sich das *History Graph Model* und das *raum-zeitliche, objektorientierte Datenmodell* als relevant.

Das History Graph Model basiert auf der Unterscheidung von Attribut- bzw. Objekt-Veränderung hinsichtlich ihres zeitlichen Verlaufs (Renolen, 1997, Kap. 4). Innerhalb des Modells werden vier Kategorien an Veränderungen eingeführt. Ein Attribut bzw. Objekt verändert sich:

- durch ein Ereignis an einem Zeitpunkt (stepwise constant)
- kontinuierlich (continuously changing)
- über einer Zeitspanne hinweg (gradually changing)
- diskret und existiert im Datenbestand nur an einzelnen Zeitpunkten (discrete values)

Diese vier Varianten zeitlicher Veränderung sind in Abbildung 2.6 dargestellt.

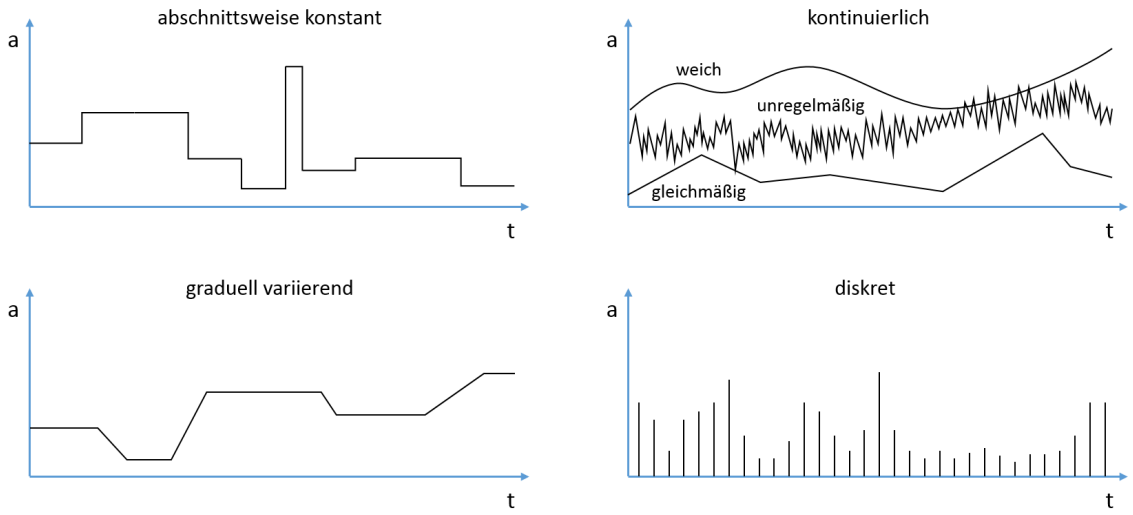


Abbildung 2.6: Zeitliche Veränderungen von Attributen, nach (Renolen, 1997)

Um Objekte raum-zeitlich zu modellieren, wird nun eine Graphstruktur aufgebaut, in der Objekte als rechteckige Knoten eingefügt werden. Die zweite Art von Knoten repräsentiert Veränderungen, die auf Objekte einwirken. Diese Knoten werden als Übergänge (engl. transitions) bezeichnet und als abgerundete Rechtecke bzw. als kreisförmige Knoten im Graphen dargestellt. Das Modell umfasst sechs Übergangsarten für die Repräsentation der Erstellung, Änderung, Zerstörung, Wiedererstellung, Aufteilung und Vereinigung von Objekten. Für ein anschauliches Beispiel der beschriebenen Modellierung sei auf (Pelekis et al., 2004, Kap. 3.6)

verwiesen. Im Kontext eines Landinformationssystems wird hier ein History Graph für eine Menge aus Parzellen und Gebäuden erstellt.

Für die Realisierung komplexer Systeme haben sich objektorientierte Methoden durchgesetzt (Booch, 1994, S. iii). Daher liegt es nahe die Einsatzeignung einer *objektorientierte Modellierung* für (raum-)zeitliche Fragestellungen zu untersuchen.

Von Wuu & Dayal (1992, Kap. 3) wird ein zeitliches Modell für objektorientierte Datenbanken beschrieben. In diesem können Objekte zeitabhängige Attribute aufweisen. Zusätzlich wird ein spezielles Vererbungskonzept vorgestellt, so dass Objekte im zeitlichen Verlauf zwischen unterschiedlichen Subklassen einer Superklasse wechseln können. Auf die Vereinigung räumlicher und zeitlicher Aspekte innerhalb eines einzelnen Objekts geht Worboys (1994) ein. Neben der Geometrie aggregiert der entwickelte Objekttyp *bitemporäre* Daten, die sich aus der Valid Time und der Transaction Time zusammensetzen.

Neben den genannten Methoden existiert noch eine Reihe weiterer Vorschläge zur objektorientierten raum-zeitlichen Modellierung. Einen Überblick liefert Pelekis et al. (2004, Kap. 3.9). Mit der *ISO-19108* ist außerdem ein internationaler Standard verfügbar, der ein objektorientiertes, konzeptionelles Modell zur Integration der zeitlichen Dimension in Geoinformationssysteme beschreibt (ISO, 2002b). Hierfür werden zeitliche Attribute, Operationen, Metadaten und mögliche Assoziationen zwischen Objekten definiert sowie auf zeitliche Bezugssysteme und deren Komponenten eingegangen.

Die aufgezeigten Modellierungen raum-zeitlicher Systeme der Geoinformatik basieren auf einer Datenerfassung in der realen Welt. Dadurch ergibt sich das Konzept zweier zeitlicher Dimensionen innerhalb dieser Systeme. Die Valid Time spiegelt die Gültigkeit von Daten wieder, die Transaction Time den zeitlichen Verlauf von Einfüge- und Lösch-Operationen. Die 4D-Gebäudemodellierung im Bauwesen ist hingegen in den Bereich der Planung einzuordnen. Zwar fließen in situ aufgenommene Geodaten in ein Projekt mit ein. Diese unterliegen jedoch häufig keiner zeitlichen Veränderung oder werden auf Basis des Snapshot Models aktualisiert. In Entwurfsprozessen stellt die Realisierung von Bauteilen hingegen die eigentliche zeitliche Dimension von 4D-Gebäudemodellen dar. Basierend auf einem antizipierten Realisierungszeitraum ergibt sich für jedes Bauteil ein graduell steigender Wert bezüglich dessen Fertigstellungsgrads.

Auf die integrierte zeitliche Modellierung in Gebäudemodellen wird in Abschnitt 3.5 des nachfolgenden Kapitels näher eingegangen. Der nächste Abschnitt behandelt die computergestützte Repräsentation von Stadtmodellen.

2.3 Datenmodellierung mit CityGML

Für eine Vielzahl deutscher Großstädte existieren bereits detaillierte Modelle der urbanen Bebauung. Aus dieser Datenbasis lassen sich computergestützt 3D-Visualisierungen erstellen, die beispielsweise die Integration projektierter Bauvorhaben in den Bestand zeigen. Die Visualisierung und geometrische Repräsentation ist jedoch nur ein Anwendungsfall dreidimensionaler, digitaler Stadtmodelle. Angestrebt wird ein Datenmodell, das Simulationen, Analysen und Bewertungen unterstützt. Hierzu zählen beispielsweise räumliche Fragestellungen bezüglich Lärmbelastungen, Netzausbau und Katastrophenschutz (Kolbe, 2009). Um diesen Anforderungen gerecht zu werden, bedarf es Konsens hinsichtlich der syntaktischen und der semantischen Modellierung dieser Modelle (Bishr, 1998).

Seit 2002 wird diesbezüglich das Datenmodell CityGML entwickelt, das sowohl geometrische als auch semantische Aspekte städtischer Bebauung repräsentieren kann. Inzwischen ist CityGML als Implementierungsspezifikation der OGC in der Version 2.0 verfügbar und realisiert ein offenes, XML-basiertes Austauschformat für 3D-Stadtmodelle (OGC, 2012). Dem Namen entsprechend basiert CityGML auf der allgemeineren *Geography Markup Language (GML)*.

Unter Bezugnahme auf Standards der ISO-19100-Familie, stellt GML ein generelles XML-basiertes Vokabular bereit, um geographische Inhalte auf Featureebene zu beschreiben. Um das übergeordnete Datenmodell nutzen und erweitern zu können, ist CityGML als Anwendungsschema der GML definiert (OGC, 2012, S. 12). Die Definition eines Anwendungsschemas erlaubt es, domänenspezifische Feature-Klassen und deren Semantik festzulegen. Zum Zeitpunkt der Drucklegung dieser Arbeit liegt CityGML in Version 2.0 vor und nutzt GML Version 3.1.1.

Neben einem Kernmodul beinhaltet CityGML dazu dreizehn thematische Module, die unter anderem die Bereiche Gebäude, Brücken, Tunnel, Gelände, Transportwege und Vegetation behandeln. In Tabelle 2.2 sind diese Module aufgeführt und deren Anwendungsbereiche beschrieben.

Die Grundlage von CityGML wird in dessen Kernmodul definiert, indem abstrakte Basisklassen und ihre Vererbungs- und Aggregationsbeziehungen beschrieben werden. Außerdem werden im Kernmodul nicht-abstrakte Klassen, die in unterschiedlichen thematischen Modulen zur Anwendung kommen, gebündelt. Als Basis einer objektorientierten, thematischen Modellierung in CityGML dient die GML-Klasse _Feature. Deren XML-Schemadefinition ist in Auflistung 2.1 dargestellt.

Durch die Eigenschaft *boundedBy*, die eine Instanz der Envelope-Klasse referenziert, kann jedem Feature eine zwei- oder dreidimensionale, umschließende Hüllgeometrie zugewiesen werden. Die *location*-Eigenschaft eines Features drückt dessen Position aus. Die _Feature-Klasse

Modul	Beschreibung des Anwendungsbereichs
Appearance	Oberflächen-Texturierung von Featuren
Bridge	Brücken und deren Bestandteile
Building	Gebäude und deren Bestandteile
CityFurniture	Unbewegliche Stadt-Einrichtungen
CityObjectGroup	Benutzerdefinierte Feature-Aggregationen
Generics	Zusätzliche Feature-Klassen und attributive Erweiterungen bestehender Klassen
LandUse	Klassifizierung der Bodennutzung
Relief	Gelände
Transportation	Transportwege
Tunnel	Tunnel und deren Bestandteile
Vegetation	Vegetation
WaterBody	Gewässer
TexturedSurface (veraltet)	Visualisierungsparametern, ersetzt durch Appearance

Tabelle 2.2: Die dreizehn Module zur thematischen Modellierung in CityGML 2.0

ist Teil der GML-Klassenhierarchie. In Abbildung 2.7 ist ein Ausschnitt aus dieser Klassenhierarchie dargestellt. Die gezeigten Klassen werden in Tabelle 2.3 näher beschrieben.

Zur geometrischen Feature-Repräsentation werden in CityGML von _Geometry ableitende Klasse genutzt. So können ein- bis dreidimensionale Geometrie beispielsweise mit Point, _Curve, _Surface und _Solid realisiert werden. Des Weiteren werden drei Arten zusammengesetzter Geometrieobjekte unterstützt (OGC, 2012, S. 20). Aggregates wie MultiPoint und MultiSurface beherbergen mehrere Primitive der gleichen Dimensionen, wohingegen ein Complex aus unterschiedlichen Typen bestehen kann. Die einzelnen Teile in einem Komplex dürfen sich dabei

```
<element name="_Feature" type="gml:AbstractFeatureType" abstract="true"
substitutionGroup="gml:_GML"/>
<complexType name="AbstractFeatureType" abstract="true">
  <complexContent>
    <extension base="gml:AbstractGMLType">
      <sequence>
        <element ref="gml:boundedBy" minOccurs="0"/>
        <element ref="gml:location" minOccurs="0"/>
        <!-- adnl. properties must be specified in an application schema -->
      </sequence>
    </extension>
  </complexContent>
</complexType
```

Auflistung 2.1: Die Definition des _Feature-Typs, Ausschnitt aus dem GML-XSD-Schema

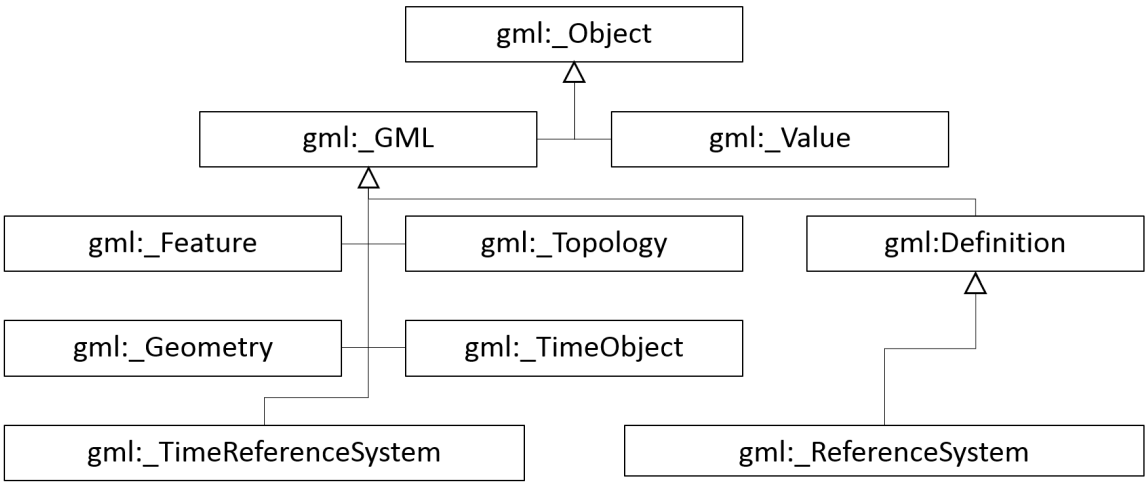


Abbildung 2.7: Stark vereinfachtes UML-Klassendiagramm des GML-Datenmodells, nach (Simon et al., 2004, S. 13)

nicht überlappen, sondern lediglich berühren. In einem Composite wie beispielsweise CompositeSurface können wiederum nur zusammenhängende Primitive mit der gleichen Dimension aufgenommen werden.

Topologische Aspekte können in CityGML durch die Verwendung von XML-basierten Verweisen, so genannten *XLinks* ausgedrückt werden. Damit nutzt CityGML nicht das von GML bereitgestellte topologische Modell der ISO-19107, sondern eine vereinfachte, an das Anwendungsschema angepasste Form (Kolbe, 2009, Kap. 2.4). Hier wird auf die explizite Modellierung mit topologischen Instanzen verzichtet (vgl. Abbildung 2.4). Folgende topologische Beziehungen können jedoch in CityGML abgebildet werden:

- Zwei unterschiedliche Feature können auf dieselbe Geometrie verweisen.
- Die Geometrie eines Features kann durch eine zweite, erweiterte Geometrie als Teilrepräsentation genutzt werden.
- Die explizit vorgehaltene Berührungsgeometrie zwischen zwei Featuren wird von beiden Instanzen genutzt.

Bezüglich des letztgenannten Falls zeigt Abbildung 2.8 ein Gebäude und einen Anbau. Beide dreidimensionalen räumlichen Körper referenzieren dabei eine einzelne Grenzfläche. Auflistung 2.2 demonstriert die entsprechende XML-basierte Modellierung.

Wie bereits erwähnt, basiert die thematische Modellierung in CityGML auf dreizehn separaten Modulen und einem zentralen Kernmodul. Innerhalb des Kernmoduls wird _CityObject als Subklasse von _Feature definiert. Die abstrakte _CityObject-Klasse dient wiederum spezialisierten Featuren wie Building und Tunnel als Basisklasse. Diesbezüglich zeigt Abbildung 2.9 eine Auszug aus dem UML-Klassendiagramm des Basismoduls.

GML Klasse Element	Beschreibung des Anwendungsbereichs
_Object	Untypisiertes XML-Element
_GML	Objekte mit Identität
_Value	Skalarer Werte, zusammengesetzte Werte und Wertebereiche
_Feature	Feature-Modellierung (ISO-19109)
_Geometry	Geometrische Modellierung (ISO-19107)
_Topology	Topologische Modellierung (ISO-19107)
_TimeObject	Zeitpunkte und Zeitperioden (ISO-19108)
_TimeReferenceSystem	Zeitreferenzsysteme (ISO-19018)
Definition	Definitionen, beispielsweise Maßeinheiten
_Coverage	Coverage-Modellierung (ISO-19123)
_ReferenceSystem	Spezialisierung konkreter Referenzsysteme (ISO-19111)

Tabelle 2.3: Beschreibung zentraler GML-Klassen

```
<bldg:BuildingPart>...
  <bldg:lod2Solid>...
    <gml:Polygon gml:id="wallSurface4711">...</gml:Polygon>
  </bldg:lod2Solid>...
</bldg:BuildingPart>

<bldg:BuildingPart>...
  <bldg:lod2Solid>...
    <gml:surfaceMember xlink:href="#wallSurface4711"/>...
  </bldg:lod2Solid>...
</bldg:BuildingPart>
```

Auflistung 2.2: XML-basierte topologische Modellierung, Ausschnitt aus einer CityGML-Instanzdatei, aus Kolbe (2009, S. 11)

Durch die neu hinzukommenden Attribute in einem _CityObject können dessen relative Lage zum Gelände bzw. zur Wasseroberfläche sowie zeitliche Daten zu dessen Entstehung und Untergang einbezogen werden. Von einem _CityObject kann außerdem auf externe Ressourcen außerhalb der eigentlichen Instanzdatei verwiesen werden.

Mit Hilfe des *generalizesTo*-Attributs der _CityObject-Klasse kann eine Objekthierarchie aufgebaut werden, die ein Feature in mehreren Detaillierungsstufen repräsentiert. CityGML realisiert dementsprechend das aus der Kartographie bekannte Vorgehen, Features anwendungsspezifisch in unterschiedlichen Detaillierungsstufen abzubilden (OGC, 2012, Kap. 6). Eine derartige Stufe wird als *Level of Detaill (LoD)* bezeichnet. CityGML definiert LoD0 bis LoD4. Abbildung 2.10 zeigt ein Gebäude in allen fünf Detaillierungsstufen.

In jeder nachfolgenden Stufe erhöht sich die geometrische Detaillierung und Genauigkeit als auch die thematische Differenzierung eines Features bzw. einer Featuregruppe. Dadurch können in CityGML Datenquellen mit unterschiedlichen Genauigkeiten in einem Modell kombi-

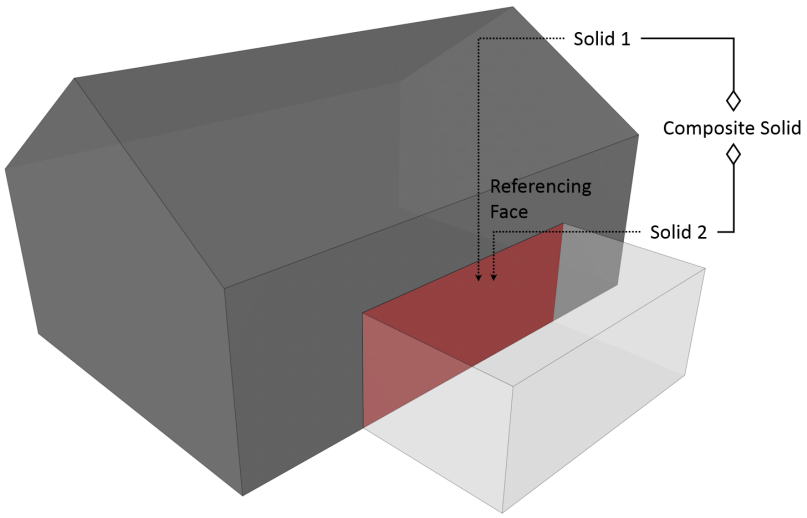


Abbildung 2.8: Topologische Adjazenz zweier dreidimensionaler räumlicher Körper in CityGML, nach (Kolbe, 2009, S. 10)

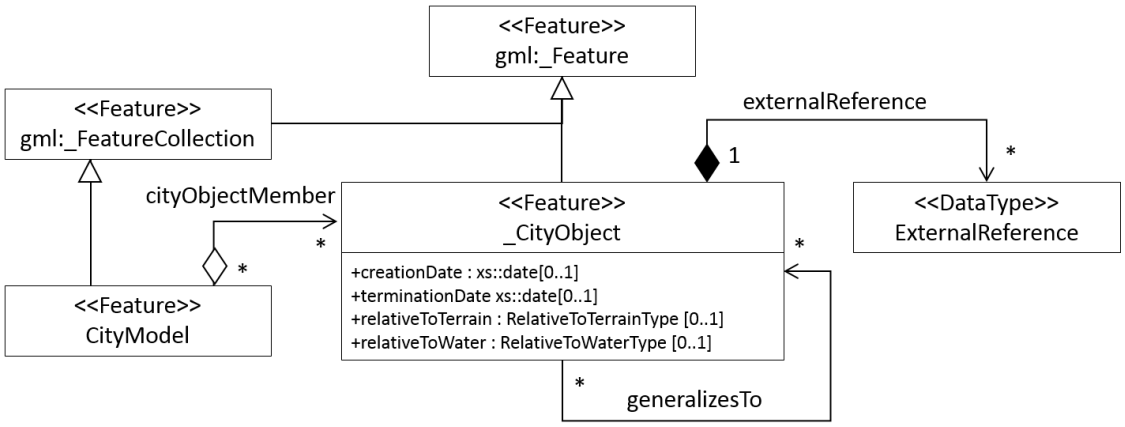


Abbildung 2.9: Auszug aus dem UML-Klassendiagramm des CityGML-Basismoduls, nach (OGC, 2012, S. 50), vereinfacht

niert werden. Außerdem ist es durch die unterschiedlichen Detaillierungsstufen möglich, für nachgelagerte Verarbeitungsschritte die geeignetste Datengrundlage auszuwählen (Kolbe & Gröger, 2004, Kap. 3).

Auf Grund der thematischen Nähe zum Building Information Modeling wird im Folgenden das Building Module vorgestellt. Im Anhang ist dazu das komplette UML-Diagramm des Moduls abgebildet (siehe Abbildung A.1). Als Ableitung von _CityObject wird hier _AbstractBuilding definiert, das als Basisklasse für Gebäude (Building) und Gebäudeteile (BuildingPart) dient. Zur seiner geometrischen Beschreibung referenziert ein Gebäude Volumenkörper (_Solid) oder Flächenmodelle (MultiSurface) in LoD1 bis LoD4. Die Hüllgeometrie eines Gebäudes kann ab LoD2 feingranular modelliert werden. Die _BoundarySurface-Klasse wird dazu in eine Vielzahl

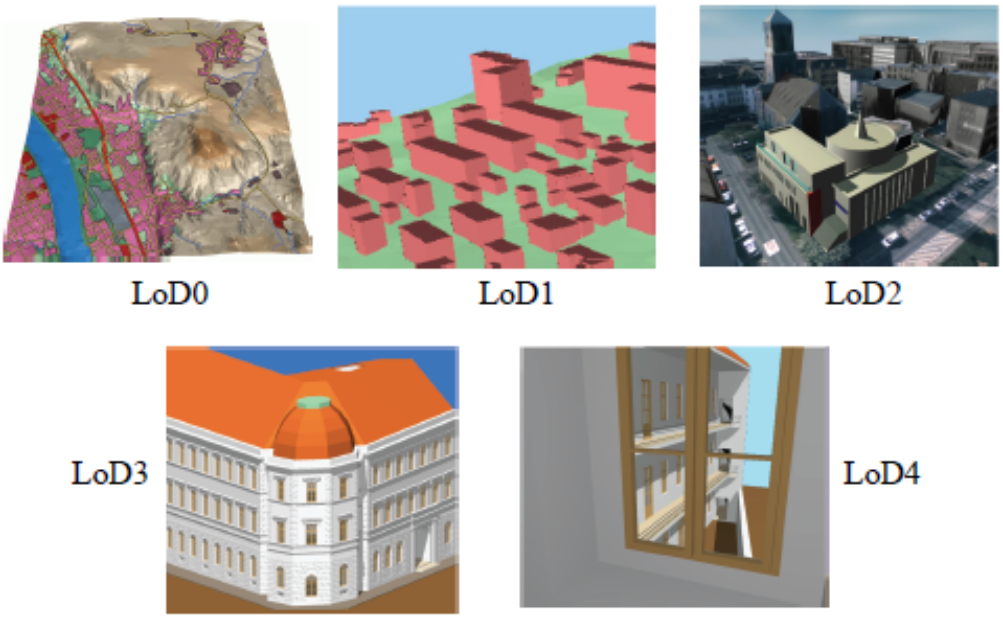


Abbildung 2.10: Detaillierungsstufen LoD0 bis LoD4 in CityGML, aus (OGC, 2012, S. 11)

an Unterklassen wie RoofSurface, WallSurface, InteriorWallSurface und FloorSurface spezialisiert.

Abbildung 2.11 zeigt diesbezüglich eine beispielhafte flächenbasierte Modellierung. Ab LoD3 können dem Modell Öffnungen, die Fenster oder Türen beherbergen, hinzugefügt werden. In LoD4 kann ein _AbstractBuilding außerdem Räume mit entsprechender Geometrie referenzieren. Zur Repräsentation der äußeren und inneren Installationen eines Gebäudes dienen die Klassen BuildingInstallation und IntBuildingInstallation.

Mit CityGML 2.0 ist es möglich, Stadtmodelle mit hohem semantischen und geometrischen Informationsgehalt bereitzustellen. Aus Sicht des Bauwesens kann eine derartige Datenbasis vorteilhaft mit BIM-basierten Methoden verknüpft werden. So erlaubt die Kombination von BIM- und CityGML-Daten die Durchführung von Planungsaufgaben, die gleichzeitig klein- und großskaliger Betrachtungen bedürfen (vgl. Kap. 8.1.4).

Im nächsten Abschnitt wird auf die grundlegenden Analysefunktionalitäten in Geoinformationssystemen eingegangen.

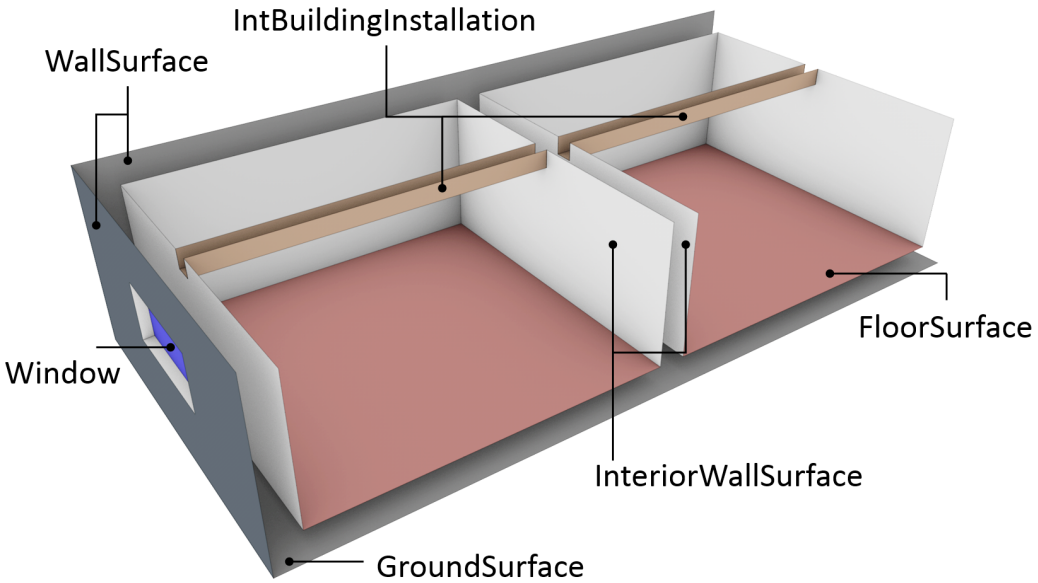


Abbildung 2.11: Flächenbasierte geometrische Modellierung in CityGML, nach (Nagel et al., 2009, S. 5)

2.4 Analysen in Geoinformationssystemen

In Abschnitt 2.1 wurde erläutert, dass Geoinformationssysteme Funktionen zur Erfassung, Verarbeitung, Analyse und Präsentation von Geodaten bereitstellen. In Abbildung 2.12 wird das Zusammenspiel dieser Komponenten verdeutlicht.

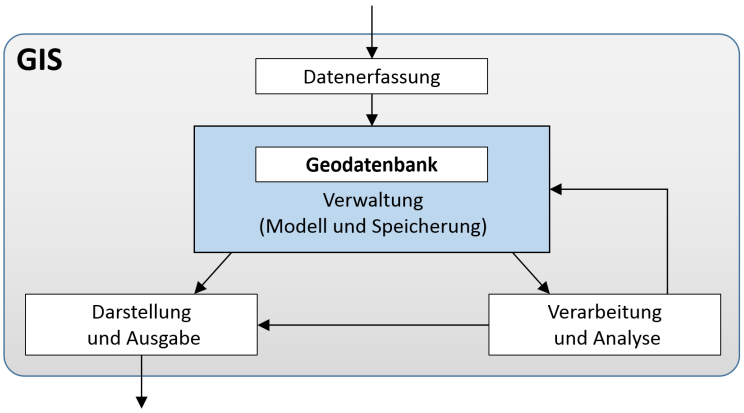


Abbildung 2.12: Die Komponenten eines Geoinformationssystems, nach (Brinkhoff, 2013, S. 3)

Durch die Analysefunktionen kann die Datenbasis ausgewertet und nach benutzerdefinierten Vorgaben prozessiert werden. Dabei fließen die entstehenden neuen Daten wiederum in den Bestand ein. Durch die aufbereiteten Daten lassen sich komplexe, räumliche Fragestellungen beantworten, was wiederum die initiale, mitunter aufwendige Datenerfassung rechtfertigt. Um derartige räumlich-semantische Analysen durchzuführen, bedarf es Operationen, die Geodaten

verarbeiten können. Eine grobe Einteilung sieht die Kategorisierung in logische, arithmetische, statistische, geometrische und topologische Operationen sowie Kombinationen aus diesen vor (Bernhardsen & Viak, 2002, Kap. 14.1).

Im Folgenden wird auf ausgewählte Grundfunktionen in Geoinformationssystemen eingegangen. Sie dienen als Basis zur Realisierung von metrischen, direktionalen und topologischen Operatoren innerhalb der entwickelten Anfragesprache für Gebäudemodelle.

2.4.1 Räumliche Grundfunktionen in Geoinformationssystemen

Geoinformationssysteme stellen Grundfunktionen zur Berechnung räumlicher Eigenschaften von Objekten bereit. Diese Eigenschaften können einem Objekt zugeordnet sein oder zwischen zwei Objekten gelten. Beispiele für Objekteigenschaften sind Umfang, Fläche und Volumen. Zwischen zwei räumlichen Objekten ergeben sich Eigenschaften wie Abstand und Winkel sowie topologische Prädikate.

Abbildung 2.13 und die folgenden beispielhaften Anfragen veranschaulichen die Anwendung der räumlichen Grundfunktionen innerhalb eines zweidimensionalen GIS im Kontext von Katasterdaten:

- Welche Bebauungen befinden sich innerhalb von Flurstück 728?
- In welchem Abstand liegen Bauwerk 56 und 58 zueinander?
- Befinden sich Bebauungen innerhalb von 5 Metern Abstand zu Bauwerk 56?

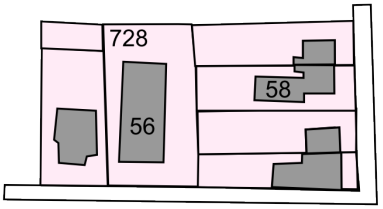


Abbildung 2.13: Beispielhafte Flurkarte

Im Rahmen dieser Arbeit sind speziell die *euklidische Abstandsberechnung* in einem metrischen Raum, der *Bounding Box Test*, die *Innen/Außen-Betrachtung* und die *Puffergenerierung* von Interesse, da diese Funktionen in angepasster Form in der entwickelten Anfragesprache zur Anwendung kommen (vgl. Kap. 5.2).

Euklidische Abstandsberechnung

Lange (2013, Kap. 4.2.1) definiert einen metrischen Raum folgendermaßen:

> Ein metrischer Raum besteht aus einer Menge M und einer Abstandsfunktion $d(a, b)$. Für zwei Elemente a und b aus der Menge M müssen folgende Eigenschaften gelten:
>
> 1. $d(a, b) \geq 0$ für alle $a, b \in M$

2. $d(a,b) = d(b,a)$
3. $d(a,b) \leq d(a,c) + d(c,b)$

Für die euklidische Metrik gilt folgende Abstandsfunktion d wobei n die Anzahl der Raumdimensionen darstellt:

$d_n(\overrightarrow{X_i}, \overrightarrow{X_j}) = \sqrt{\sum_n^{k=1}(x_{ik} - x_{jk})^2}$ mit $\overrightarrow{X_i} = (x_{i1}, x_{i2}, ...x_{in})$ und $\overrightarrow{X_j} = (x_{j1}, x_{j2}, ...x_{jn})$

Dadurch ist der euklidische Abstand zwischen Punkten definiert. Bei dreidimensionalen Körpern sind sowohl der minimale als auch der maximale Abstand zwischen diesen erfassbar. Als Eigenschaft eines dreidimensionalen Körpers ist außerdem der maximale Abstand zweier Punkte auf diesem von Interesse.

Bounding Box Test

Geometrische Algorithmen in Kombination mit ausgedehnten Datenbeständen können zu rechenintensiven Operationen führen und damit die Reaktionsfähigkeit eines Informationssystems negativ beeinflussen. Dem kann durch effiziente Vortests entgegengewirkt werden, wodurch aufwendigere Berechnungen seltener durchgeführt werden mussen.

Eine diesbezügliche Grundfunktion ist der Bounding Box Test (Bill, 2010, Kap. 7.2.2). Interessiert die Interaktion zwischen Featuren, wie beispielsweise ob sich deren geometrische Repräsentationen schneiden, wird im ersten Schritt nicht deren tatsächliche Geometrie untersucht. Stattdessen werden im zweidimensionalen Fall minimal umschließende Rechtecke um die Objekte gebildet. Diese Approximation ist größer oder gleich der eigentlichen Geometrie, weswegen gilt: Sind diese Rechtecke zweier 2D-Feature disjunkt, besteht auch keine Interaktion zwischen den detaillierten Repräsentationen. Im dreidimensionalen Fall werden Körper mit einer minimal umschließenden Box-Geometrie eingefasst. Die englische Bezeichnung *Axis Aligned Bounding Box (AABB)* verdeutlicht zudem, dass die Box-Geometrie an das verwendete Koordinatensystem ausgerichtet ist. Dies vereinfacht die Erstellung derartiger umschließender Geometrie und führt zu effizienten Schnitttests.

Indizierungsstrukturen, wie der *R-Tree* und der *Octree*, nutzen den AABB-Test als Grundfunktion, um während räumlicher Suchen frühzeitig Kandidaten aus einer Hierarchie ausschließen zu können (siehe nachfolgende Abschnitte 2.5.1 und 2.5.2).

Des Weiteren besteht die Möglichkeit eine an die jeweilige Geometrie ausgerichtete Box für Vortests zu nutzen. Diese wird im Englischen als *Oriented Bounding Box (OBB)* bezeichnet. Im allgemeinen Fall verringert sich durch die Verwendung einer OBB der Leerraum zwischen Box und eigentlichem Körper. Von Gottschalk et al. (1996) werden die entsprechenden Erstellungsmethoden und Schnitttests vorgestellt.

Punkt in Polygon Test

Die Frage, ob sich ein Punkt innerhalb eines beliebigen, geschlossenen Polygons befindet, stellt eine grundlegende, topologische GIS-Operation dar. Das Polygon kann dabei eine konvexe oder konkave Form annehmen. Es darf jedoch keine Selbstüberschneidungen beinhalten und muss demnach eine einfache Geometrie darstellen. Huang & Shih (1997) betrachten acht Algorithmen zur Lösung des Punkt-in-Polygon-Tests, sowohl für Raster- als auch für Vektor-Daten und vergleichen deren Komplexität. Darunter der *Ray Intersection*-Algorithmus, der auf dem *Theorem von Jordan* beruht.

Das Theorem besagt, dass jedes geschlossene Polygon die Trennung der Ebene in die zwei Bereiche Innen und Außen bewirkt (Trybulec et al., 2007, Kap. 2). Daher muss ein Strahl, der vom Inneren des Polygons ausgesandt wird, die Grenze zwischen Innen und Außen in einer ungeraden Anzahl von Schnitten passieren. Hingegen ist die Schnittanzahl bei der Aussendung des Strahls von außerhalb des Polygons gerade (siehe Abbildung 2.14).

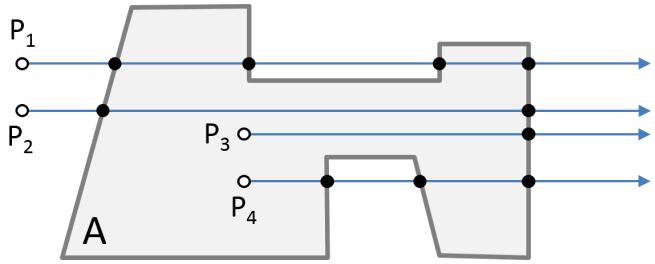


Abbildung 2.14: Außen/Innen-Verifikation zwischen den Punkten P_{1-4} und Polygon A

Mögliche Sonderfälle sind Schnitte des Strahls mit dem Eckpunkt des Polygons und schleifende Schnitte, wenn der Strahl auf einer Kante des Polygons aufliegt. Um die Robustheit der Methode zu erhöhen, werden daher mehrere Strahlen pro Test genutzt. Die Zeitkomplexität des Algorithmus entspricht $\mathcal{O}(n)$, wobei n die Anzahl der Kantenstücke des Polygons angibt.

Eine dreidimensionale Variante des Ansatzes in Kombination mit einer räumlichen Indexstruktur wird in der Laufzeitumgebung der entwickelten Anfragesprache eingesetzt. Die Methode erlaubt die effiziente Auswertung topologischer Relationen und wird in Kapitel 5.2.3 vorgestellt. Für eine Beschreibung der weiteren Verfahren, die unter anderem nur konvexe Polygone unterstützten (*Sum of Area*-Algorithmus) oder anfälliger für Rundungsfehler sein können (*Sum of Angles*-Algorithmus), sei auf Huang & Shih (1997) verwiesen.

Pufferzonen

Eine weitere geometrische Grundfunktion ist die Generierung von sogenannten Pufferzonen, die mit einem benutzerdefinierten Abstand um ein Feature gebildet werden. Andere Feature

können auf Überlappung oder Inklusion gegenüber dieser Puffergeometrie getestet werden. Konzeptionell ergibt sich dadurch die Möglichkeit einer qualitativen Bestimmung der Nähe zwischen Featuren. Dementsprechend können beispielsweise alle Entitäten, auf die sich ein räumlich begrenztes Ereignis auswirkt, auf Basis eines Puffers selektiert werden.

In Geoinformationssystemen wird standardmäßig die Puffererstellung um Punkten, Linien und Polygonen unterstützt. Puffer können als Kreispuffer oder als Quadratpuffer realisiert werden. Abbildung 2.15 zeigt die zwei Varianten der Generierung mit den unterschiedlichen Ausgangsobjekten Punkt, Linie und Polygon.

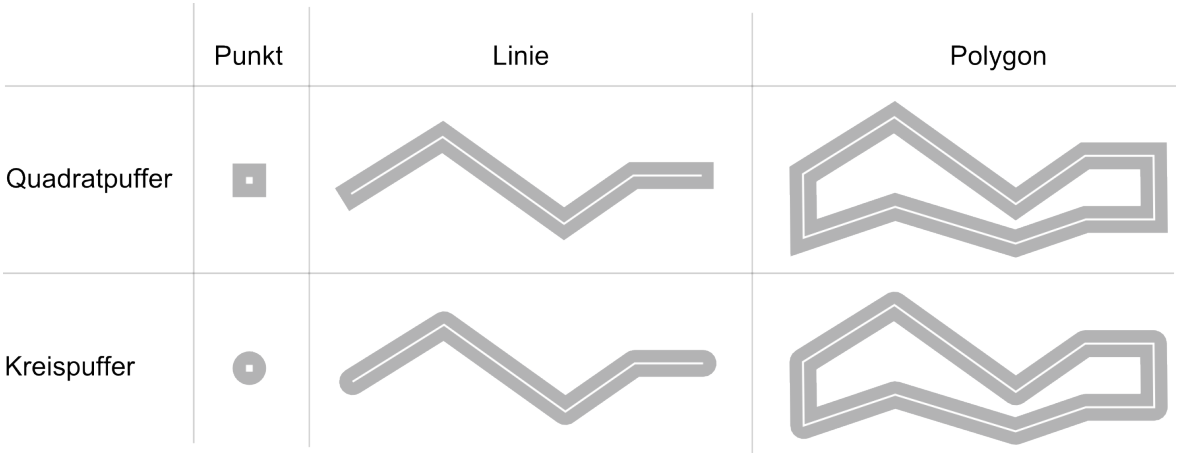


Abbildung 2.15: Quadrat- und Kreispuffer um Punkten, Linien und Polygonen

Im Falle von Punkten als Ausgangsgeometrie ist die Erstellung der Puffergeometrie trivial. Die geometrische Berechnung der Puffergeometrie für Linien und Polygon hingegen ist aufwendig. Prinzipiell werden dabei Liniensegmente parallel zum Ursprungssegment verschoben. In den Schnittregionen zwischen den einzelnen Pufferteilen muss die Übergangeometrie berechnet werden. Das Auftreten von Selbstüberschneidungen muss zusätzlich berücksichtigt werden. Die Erzeugung von Puffern aus dreidimensionalen Körpern ist in Geoinformationssystemen derzeit häufig nicht möglich oder wird durch Projektion der Ausgangsgeometrie auf die Grundebene als Funktion der *Planimetrie* gelöst.

2.4.2 Analysen topologischer Relationen

Bei der Betrachtung topologischer Relationen werden euklidische Distanzen außer Acht gelassen. Stattdessen wird die relative Lage räumlicher Objekte zueinander unter qualitativen Gesichtspunkten ausgewertet. In einem GIS stellt der Raumbezug das primäre verknüpfende Merkmal aller vorgehaltenen Entitäten dar. Aus diesem Grund erweisen sich topologische Relationen als effiziente Auswahlkriterien für Analysefunktionen und weiterverarbeitende Prozesse. Dabei ist der hohe Abstraktionsgrad dieser Beziehungsart der menschlichen qualitativen Auffassungsgabe ähnlich (Knauff et al., 1997, S. 5). Frühzeitig ergab sich hieraus das Be-

streben, topologische Operatoren für Geoinformationssysteme zu standardisieren. Ohne eine eindeutige formale Definition der topologischen Beziehungen ist dies jedoch nicht möglich.

Als Qualitätsmerkmale einer Definition für topologische Relationen werden von Clementini & Di Felice (1996, Kap. 1) folgende Punkte angeführt: Die Relationen müssen sich gegenseitig ausschließen, so dass ein untersuchtes Objektpaar exakt zugeordnet werden kann. Die Definition soll es außerdem ermöglichen, alle tatsächlich auftretenden Relationsarten abzubilden. Dazu bedarf es einer zugrundeliegenden Definition der beteiligten räumlichen Objekte.

Im Kontext topologischer Relationen wurden am häufigsten Punkte, Linien und Regionen in der Ebene untersucht. In (Borrmann, 2007, Kap. 6) werden innerhalb der Entwicklung einer räumlichen Algebra dreidimensionale Körper in die Betrachtung mitaufgenommen. Neben der dimensionalen Einteilung räumlicher Objekte unterscheidet man zwischen einfachen und komplexen Typen (Clementini & Di Felice, 1996, Kap. 2). Im Falle einfacher Typen ist keine Separation des Inneren und des Äußeren erlaubt, wohingegen dies bei komplexen Typen möglich ist. Ein topologisch komplexer Linientyp kann beispielsweise mehrere nichtzusammenhängende Liniensegmente beinhalten (Borrmann, 2007, Kap. 7.2.4). Abbildung 2.16 zeigt Instanzen einfacher und komplexer räumlicher Typen.

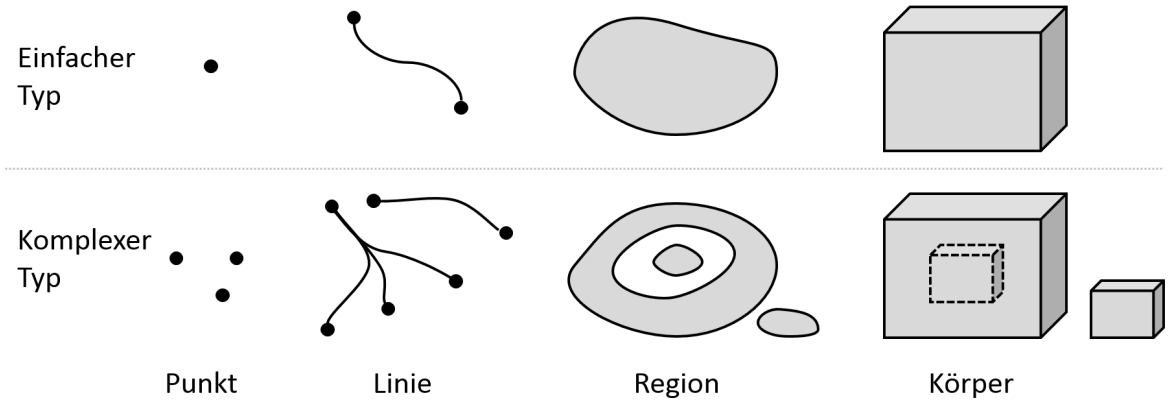


Abbildung 2.16: Beispiele einfacher und komplexer räumlicher Objekte, nach (Borrmann, 2007, S. 89)

Die im Rahmen dieser Arbeit genutzte Definition topologischer Relationen nach Egenhofer basiert auf der *Punktmengentheorie*, der *Punktmengentopologie*, der *simplizidalen Topologie* und der *zellulären Topologie*. Die Punktmengentheorie nutzt zur Repräsentation geometrischer Objekte das Konzept der mathematischen Mengen (Gaal, 1964). Dadurch lassen sich mengenbasierte Operatoren, wie Schnitt und Vereinigung, auf derartige definierte Objekte anwenden.

Von Güting (1988) werden die Relationen *equal*, *not equal*, *outside* und *intersect* für Punktmengen definiert. Diese vier Relationen schließen sich jedoch weder aus, noch sind damit alle realisierbaren Relationen zwischen räumlichen Objekten hinreichend beschrieben. Um diese

Defizite auszugleichen, werden in der Punktmengentopologie Definitionen für das *Innere (engl. interior)*, den *Abschluss (engl. closure)* und den *Rand (engl. boundary)* von Objekten eingeführt. Die formalen Definitionen aus (Egenhofer & Franzosa, 1991, Kap. 3) sind im Anhang in Abschnitt A.2 aufgeführt.

Auf den genannten Definitionen basiert das 4-Intersection Model (4-IM) (Egenhofer & Herring, 1992). Darin werden die vier Schnittmengen zwischen dem Inneren $^\circ$ und dem Rand ∂ zweier Objekte A und B ausgewertet. Durch die drei betrachteten Grundkörper Punkt, Linie und Region ergeben sich sechs zu untersuchende Konstellationen: Region und Region, Linie und Region, Punkt und Region, Linie und Linie, Punkt und Linie sowie Punkt und Punkt. Für die vier entstehenden Ergebnismengen $A^\circ \cap B^\circ$, $A^\circ \cap \partial B$, $\partial A \cap B^\circ$ und $\partial A \cap \partial B$ liefert die Schnittoperation jeweils die leere Menge $\emptyset$ oder die nicht leere Menge $\neg\emptyset$. Theoretisch ergeben sich für das 4-IM 16 Relationen, die aber nicht alle realisierbar sind. Von Egenhofer & Franzosa (1991) wurde bewiesen, dass zwischen einfachen, polygonalen Regionen in der Ebene nur acht dieser Relationen tatsächlich auftreten können (Abbildung 2.17).

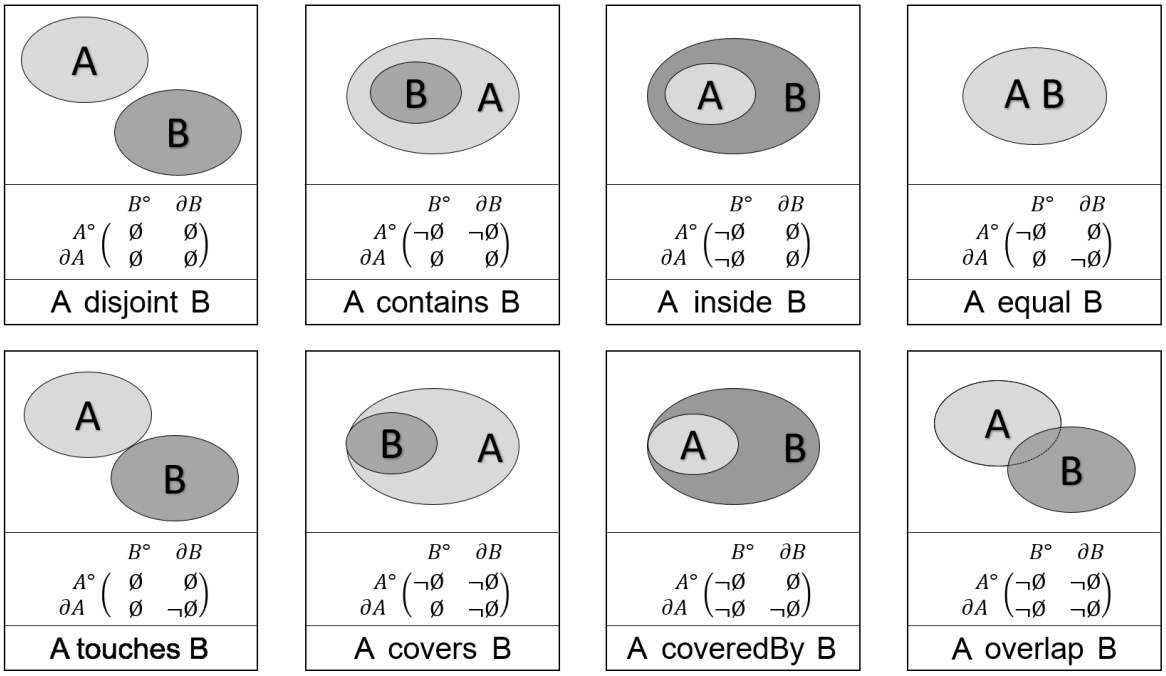


Abbildung 2.17: Das 4-Intersection Model der acht realisierbaren topologischen Relationen zwischen Regionen im zweidimensionalen Raum, nach (Egenhofer & Herring, 1994)

Das Innere von Regionen wird durch deren Rand komplett vom Äußeren getrennt. Dies ist bei Linien nicht der Fall. Dadurch können topologische Konstellationen entstehen, die durch das 4-IM nicht unterschieden werden können. Abbildung 2.18 zeigt Beispiele, in denen eine Region und eine Linie unterschiedlich miteinander interagieren, ohne dass sich dies im 4-Intersection Model widerspiegelt.

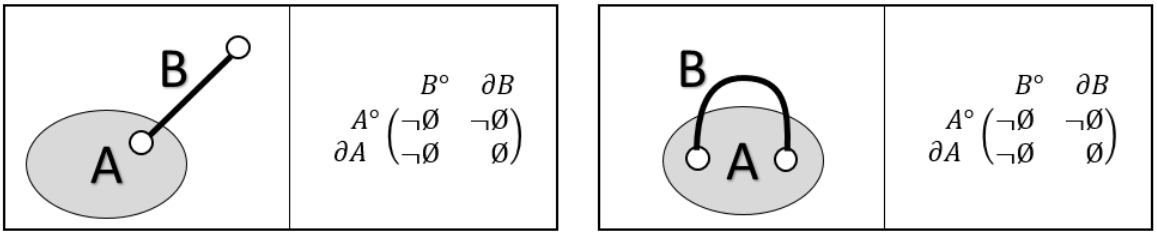


Abbildung 2.18: Ausdrucksschwäche des 4-IMs bei der Betrachtung von Linie und Regionen, nach (Egenhofer & Herring, 1992, S. 18)

Aus diesem Grund wird das 4-Intersection Model zum 9-Intersection Model (9-IM) erweitert (Egenhofer & Herring, 1992). Dabei wird das Äußere der beteiligen Objekte in die Betrachtung mit aufgenommen. Das Äußere A^- ist definiert als die Menge innerhalb des *Universums* U die nicht zum Abschluss $\overline{A}$ gehört: $A^- = U - \overline{A}$. Abbildung 2.19 zeigt das 9-Intersection Model der acht realisierbaren topologischen Relationen zwischen Regionen in der Ebene.

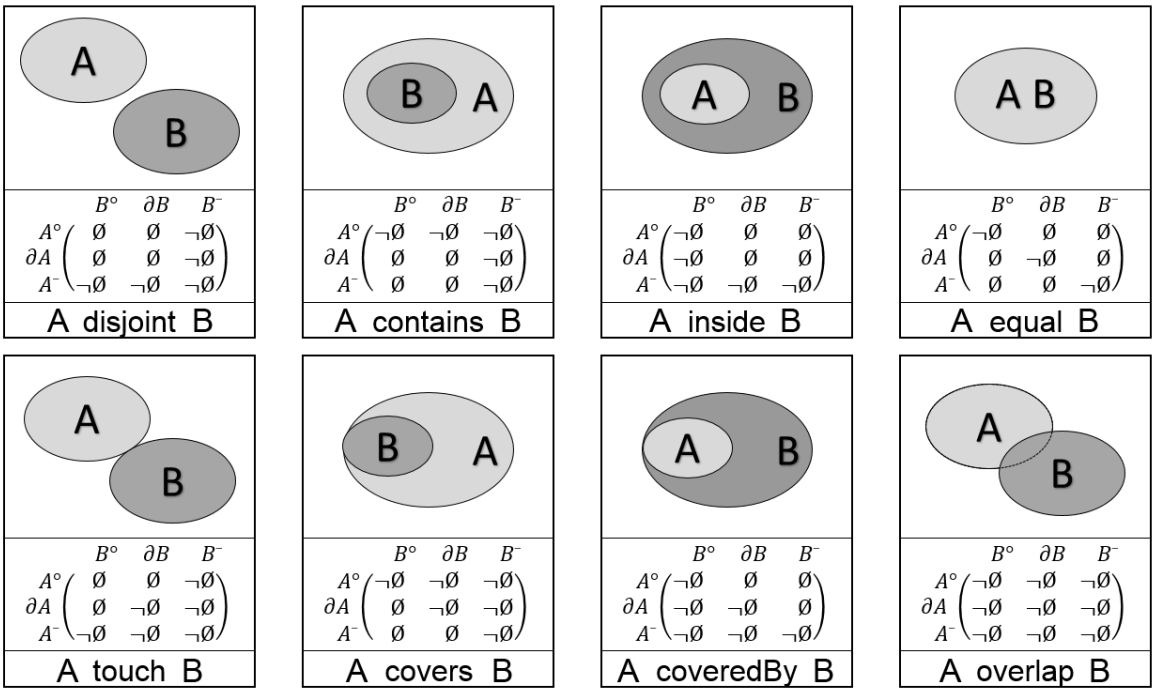


Abbildung 2.19: Das 9-Intersection Model der acht realisierbaren topologischen Relationen zwischen Regionen im zweidimensionalen Raum, nach (Egenhofer & Herring, 1994)

Mit dem *Dimension Extended 9-Intersection Model (DE9-IM)* existiert eine erweiterte Form des 9-IMs. Hier werden die Mengenverschneidungen mit der Funktion *dim*() durchgeführt. Anstelle der binären Werte $\emptyset$ und $\neg\emptyset$ gibt die Funktion -1, 0, 1 und 2 zurück. Der Wert -1 steht hierbei für die leere Menge $\emptyset$. Die weiteren Werte geben die Dimension der entstehenden Schnittmenge an. Für eine detaillierte Betrachtung des DE9-IMs sei auf (Clementini et al., 1993) und (Clementini & Di Felice, 1995) verwiesen. Innerhalb der bereits angesprochenen

Simple Feature Access wird das DE9-IM für den zweidimensionalen Fall der Planimetrie angewendet (Herring, 2011b, S. 34).

Eine Betrachtung topologischer Relationen im dreidimensionalen Raum ist in der ISO 19107 (2003) enthalten. Von Borrmann (2006) und Borrmann & Rank (2009b) wurden zudem topologische Relationen auf Problemstellungen in BIM angewendet.

In Kapitel 5.2.2 wird ein neuartiges Konzept entwickelt, um topologische Analysen für BIM-Daten effizient ausführen zu können. Hierbei müssen die steigende Komplexität im dreidimensionalen Anwendungsfall und die mitunter umfangreichen Datenbestände im BIM berücksichtigt werden.

2.5 Räumliche Indexierung

Indexstrukturen ermöglichen es, umfangreiche Datenbestände effektiv zu verarbeiten und spielen daher in der Datenbanktechnologie eine wichtige Rolle. Um beispielsweise die Frage zu beantworten, ob ein bestimmter Wert in einer Integer-Liste enthalten ist, müssen im ungünstigen Fall alle Listenelemente überprüft werden. Eine Indexstruktur für eindimensionale Daten wie der *B-Tree* ermöglicht es hingegen, die Elementsuche auch im ungünstigsten Fall in $\mathcal{O}(log(n))$ auszuführen (Bayer & McCreight, 1972). Abbildung 2.20 zeigt die Suche nach dem Integerwert 89 in einer linearen Struktur und in einem B-Tree.

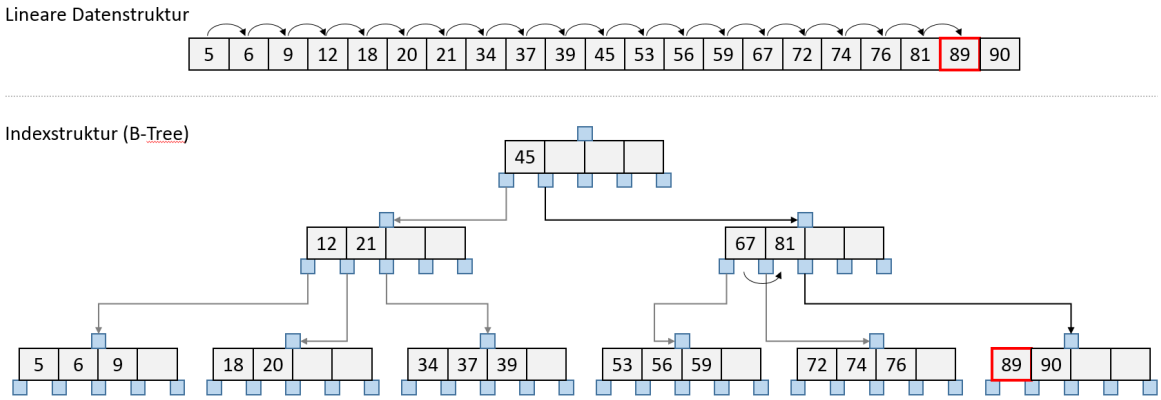


Abbildung 2.20: Eindimensionale Datenelemente in einer linearen Datenstruktur und in einem B-Tree. Die Zeitkomplexität beträgt in der linearen Struktur $\mathcal{O}(n)$ und im B-Tree $\mathcal{O}(log(n))$

Eine effiziente räumliche Selektion von Objekten ist eine entscheidende Basisfunktionalität von Geoinformationssystemen. In der Geoinformatik haben sich wegen der Mehrdimensionalität der vorgehaltenen Daten spezielle Indizierungsstrukturen etabliert. Im Folgenden werden diesbezüglich der *Quadtree*, der *R-Tree* und der *R*-Tree* vorgestellt. Indizierungsstrukturen,

wie der B-Tree und die *HashMap*, die für eindimensionale Daten Anwendung finden, werden nicht weiter erläutert. Hierfür sei beispielsweise auf (Brinkhoff, 2013, Kap. 6.1) verwiesen.

2.5.1 Die Indexstrukturen Quadtree und Octree

Ein Octree ist eine hierarchische Datenstruktur, die auf der rekursiven Teilung des indizierten, dreidimensionalen Raums in acht Quadranten basiert. Aus Gründen der Übersichtlichkeit wird die Struktur im Folgenden anhand der zweidimensionalen Variante, dem Quadtree erläutert. Innerhalb eines Quadtrees referenziert jeder Knoten entweder vier oder keine Kindknoten. Die Quadranten werden meistens als *SW*, *NW*, *SE* und *NE* entsprechend ihrer Himmelsrichtungen bezeichnet.

Es existiert eine Vielzahl an Varianten des Quadtrees. Diese ergeben sich aus den vielfältigen Geometrietypen der zu indizierenden Vektor- und Raster-Daten. Im Folgenden wird auf den *MX Quadtree*, wie er in (Hunter & Steiglitz, 1979) vorgestellt wird, näher eingegangen. Die anderen Varianten wie der *Point Quadtree*, der *Point Region Quadtree*, der *Region Quadtree* und der *Polygonal Map Quadtree* werden in (Finkel & Bentley, 1974) und (Samet & Webber, 1985) näher beschrieben.

Ein MX-Quadtree indiziert ein einfaches Polygon im zweidimensionalen Raum. Zur Erstellung des Baums wird der Raum um das betrachtete Polygon rekursiv in vier Quadranten aufgeteilt. Ist ein Quadrant komplett innerhalb oder außerhalb des Polygons lokalisiert, wird keine weitere Aufteilung durchgeführt. Die von Polygonkanten geschnittenen Quadranten werden hingegen bis zu einer benutzerdefinierten Baumtiefe rekursiv verfeinert. Wird ein Polygon mit v Stützpunkten und einem Umfang p in einem $2^n * 2^n$ großen Bereich indiziert, beträgt die Zeitkomplexität der Generierung $\mathcal{O}(v + p + n)$. Der Umfang p wird dabei als Anzahl an Quadranten der maximalen Baumtiefe gemessen.

Abbildung 2.21 zeigt beispielhaft ein Polygon und den daraus abgeleiteten Quadtree. Die Diskretisierung des Raumes ermöglicht hinsichtlich des Polygons eine Klassifizierung des Inneren, des Randes und des Äußeren. Hierbei werden die innen liegenden Quadranten schwarz und die äußeren weiß gefärbt. Bereiche, die durch das Polygon geschnitten werden, erhalten eine graue Einfärbung. Durch diese Klassifizierung liegt eine Verwendung der Baumstruktur in topologischen Auswertungen nahe (Borrmann & Rank, 2009b).

Abbildung 2.21 zeigt neben dem Quadtree auch einen Suchbereich für den alle schneidenden und enthaltenen Kanten des indizierten Polygons ausgewählt werden sollen. Dazu wird in einer *Breitensuche* nach grauen Oktanten gesucht. Die *Rekursion* stoppt auf Ebene der *Blattknoten* des Baumes. Die gefunden grauen Knoten, die in der Zeichnung gekennzeichnet sind, referenzieren ihre schneidenden Polygonkanten. Diese werden in die Ergebnismenge aufgenommen, wobei auftretende Duplikate eliminiert werden müssen. Der Vorteil dieser hierarchischen

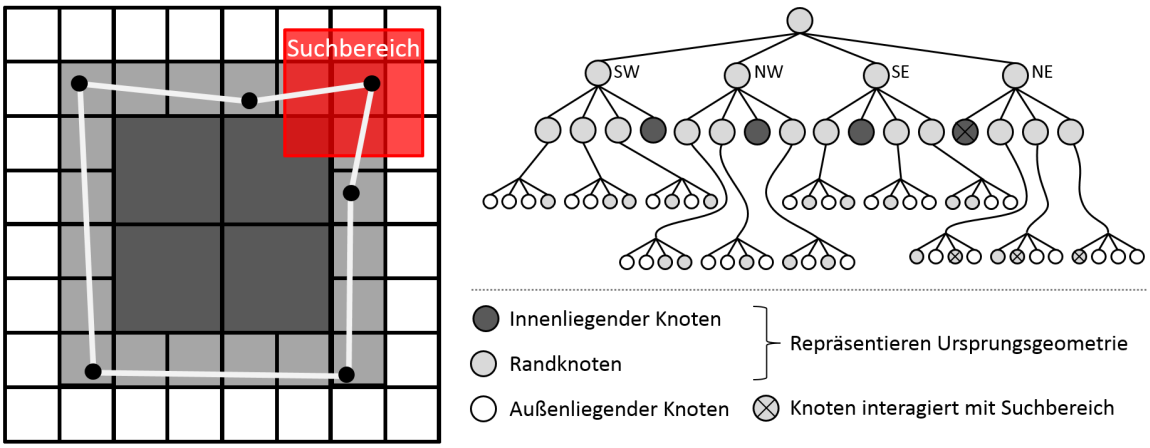


Abbildung 2.21: Ursprungsgeometrie (Polygon), abgeleitete Quadtree-Struktur (Baumtiefe = 3) und Interaktion mit einem Suchbereich

Suche ist die mitunter hohe Anzahl an frühzeitig ausschließbaren Kandidaten. Dies kann bei komplexen Geometrierepräsentationen die Suchdauer deutlich reduzieren.

2.5.2 Die Indexstruktur R-Tree

Der R-Tree ist eine dynamische Indexstruktur, in der multidimensionale Datenobjekte eingefügt, geändert und gelöscht werden können (Guttman, 1984). Aus Gründen der Übersichtlichkeit wird die Struktur anhand der Indizierung von Polygonen im zweidimensionalen Raum vorgestellt. Grundsätzlich gilt, dass jeder Knoten eine eigene AABB besitzt. Jeder referenzierte Kindknoten besitzt wiederum eine AABB, wobei die Box des Elternknotens alle Kinder-AABBs einschließt. Innerhalb der Baumstruktur sind zwei unterschiedlichen Knotenarten voneinander zu unterscheiden (siehe Abbildung 2.22). Blattknoten referenzieren Paare aus einem Datenobjekt (hier ein Polygon) und einer davon abgeleiteten AABB. Nicht-Blattknoten referenzieren keine Datenobjekte sondern weitere Knoten.

Die Anzahl der referenzierbaren Elemente in einem Knoten ist konfigurierbar. Der Parameter M des Baumes gibt hierzu die maximale Anzahl an Elementen an, die in einem Knoten dauerhaft aufgenommen werden können. Der Parameter m ist die minimal zulässige Anzahl an Elementen, die zu jeder Zeit in einem Knoten enthalten sein müssen. Zwischen M und m muss gelten: $m <= M/2$. Insgesamt weist ein R-Tree folgende Eigenschaften auf:

- Jeder Blattknoten beinhaltet zwischen m und M Elemente.
- Der Wurzelknoten hat mindestens zwei Kinder, außer er ist gleichzeitig ein Blattknoten.
- Der Baum ist balanciert, so dass alle Blattknoten auf identischer Ebene angesiedelt sind.

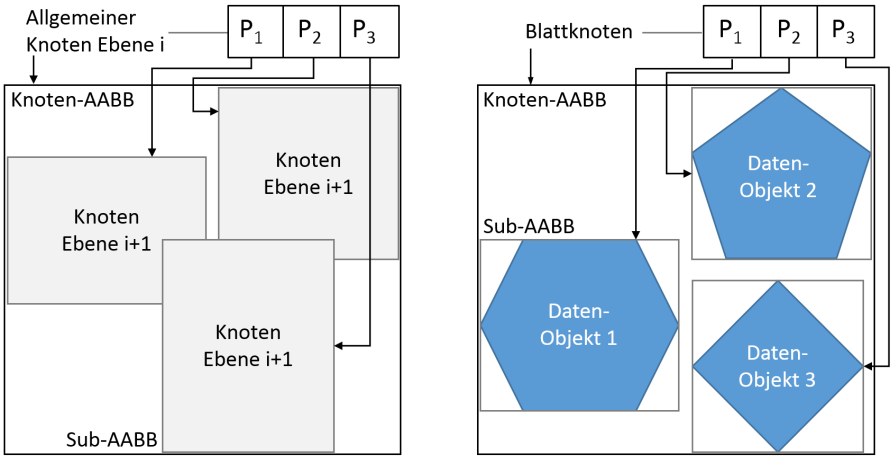


Abbildung 2.22: Aufbau der zwei Knotenvarianten des R-Trees

Durch den Parameter M ergibt sich die Möglichkeit, Knoten auf die Größe einer *Page* anzupassen. Eine Page ist die Einheit für einen virtuellen, zusammenhängenden Speicherbereich des Betriebssystems. Dieser muss gegebenenfalls von langsamen sekundären Speichermedien in den Hauptspeicher des Systems transferiert werden. Eine derartige Paging-Operation ist zeitintensiv und kann daher die Laufzeit eines Verarbeitungsschrittes negativ beeinflussen. Die Anpassung des R-Trees auf eine systemabhängige Page-Größe erlaubt die Anzahl an derartigen zeitaufwendigen Transferoperationen zu minimieren, wodurch Suchanfragen mitunter um ein Vielfaches schneller ausgeführt werden können.

Der R-Tree kann zur effizienten Suche in einem multidimensionalen Bereich genutzt werden. Zur Demonstration ist in Abbildung 2.23 ein R-Tree mit $M = 3$, $m = 1$ und der Tiefe 2 dargestellt. Dieser indiziert zwölf Polygone wobei, abgesehen von Datenobjekt 8, ausschließlich deren AABBs mit den IDs 8 bis 19 abgebildet sind. Wie in der rechts abgebildeten Baumstruktur dargestellt sind diese AABBs in den Knoten 3 bis 7 beinhaltet. In der Tiefe 1 existieren die zwei Knoten 1 und 2, die diese Blattknoten der Tiefe 2 referenzieren. Knoten 1 und 2 werden wiederum vom Wurzelknoten 0 aggregiert.

Es existieren zwei unterschiedliche Ausprägungen des Suchalgorithmus. So können ausschließliche Datenobjekte zurückgeben werden, deren AABBs komplett im Suchbereich enthalten sind. Alternativ können auch zusätzlich Datenobjekte ausgewählt werden, die den Suchbereich lediglich schneiden. In beiden Fällen wird in einer Breitensuche, ausgehend vom Wurzelknoten, jeder Knoten gegen die Suchgeometrie getestet.

Im aktuellen Beispiel liegt der rote Suchbereich innerhalb des Wurzelknoten. Ist der Suchbereich und die AABB des aktuellen Knotens nicht disjunkt, so werden die Kinderknoten als Kandidaten zwischengespeichert (Knoten 1 u. 2). Im nächsten Schritt werden die AABBs dieser Knoten gegenüber dem Suchbereich ausgewertet, wodurch Knoten 2 aus jeglicher weiteren Betrachtung ausgeschlossen werden kann. In der Tiefe 1 des Baumes interagiert dann

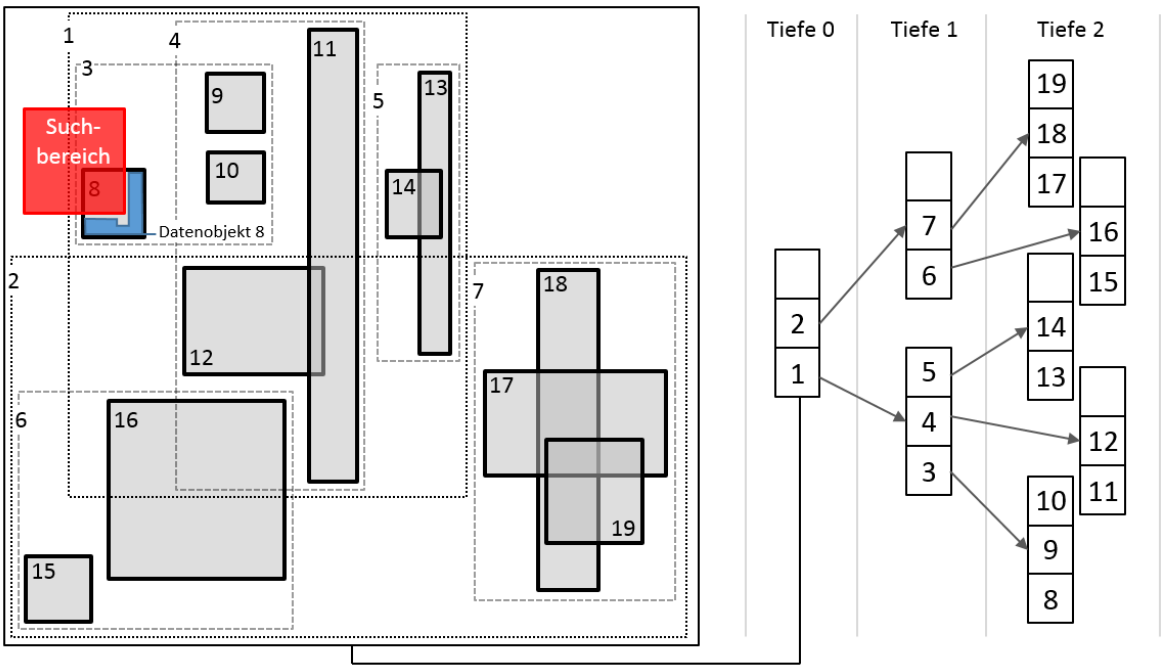


Abbildung 2.23: Beispiel für eine R-Tree-Struktur (Baumtiefe = 2) mit $M = 3$, $m = 1$ und zwölf indizierten Datenobjekten

ausschließlich Knoten 3 mit dem Suchbereich. Erreicht die Traversierung des Baumes die Ebene der Blattknoten, wird der Suchbereich mit den AABBs der indizierten Datenobjekte verglichen. Alle Datenobjekte, deren AABBs mit dem Suchbereich interagieren, werden über die enthaltenen Zeiger dereferenziert und der Ergebnismenge hinzugefügt. Hier ist dies Datenobjekt 8.

Zu beachten ist, dass die Interaktion mit dem Suchbereich lediglich auf der AABB des Datenobjekts beruht. Es werden keine Tests mit der zugrundeliegenden Ursprungsgeometrie, hier Polygon 8, ausgeführt. So dient der R-Tree für die Identifizierung von Kandidaten, die im nächsten Schritt durch aufwendigere Geometrietests verifiziert werden müssen. Entscheidend ist, dass die Auffindung der Kandidaten durch den R-Tree bei ausgedehnten Datenbeständen mit weniger Aufwand verbunden ist, als wenn gegen alle AABBs des Datenbestandes getestet wird.

Werden neue Datenobjekte zur Indizierung in den R-Baum eingefügt, wird folgendermaßen vorgegangen: (1) Ein Blattknoten, der am besten geeignet ist, muss identifiziert werden, um das Datenobjekt aufzunehmen. Dazu wird vom Wurzelknoten ausgehend jeweils der Kindknoten identifiziert, der bei Aufnahme des Datenobjekts am geringsten an Größe zunimmt. (2) Das Einfügen eines neuen Datenobjekts verändert gegebenenfalls die AABB des Blattknotens. Diese Änderung kann sich rekursiv auf Elternknoten auswirken und muss daher vom geänderten Blattknoten unter Umständen bis zum Wurzelknoten propagiert werden. (3) Falls der bestgeeignete Blattknoten bereits die maximale Anzahl M an Datenobjekte referenziert,

muss dieser sinnvoll aufgeteilt werden. (4) Wurde ein Blattknoten in zwei neue Blattknoten aufgeteilt, wirkt sich diese Änderung gegebenenfalls rekursiv auf Elternknoten aus, die durch den zusätzlichen Kindknoten ebenfalls überlaufen können. Falls der Wurzelknoten überläuft, wird auch dieser in zwei neue Knoten aufgeteilt. Ein neu erstellter Wurzelknoten nimmt dann die neuen Knoten auf, wodurch der Baum um eine Ebene in die Höhe wächst.

Läuft ein Wurzelknoten bei der Aufnahme eines Datenobjekts über, muss dieser Knoten in zwei neue Knoten aufgeteilt werden. Die Geometrie der AABBs der neuen Knoten hat direkte Auswirkung auf die Effizienz von Suchanfragen. Um möglichst wenige Kinderknoten als Kandidaten behandeln zu müssen, wird versucht, die Gesamtgröße der beiden neuen AABBs zu minimieren. Um definitiv die Aufteilung mit der kleinsten Gesamtgröße zu finden, müssten alle Kombination getestet werden. Die Anzahl an Möglichkeiten beträgt 2^{M-1}. Dies führt bei sinnvollen M-Werten, die um 50 liegen, zu einer hohen Anzahl an Operationen. Deswegen kommen heuristische Aufteilungsalgorithmen zum Einsatz, die nicht zwingend die kleinste Gesamtgröße für beide neuen AABBs finden, sondern dies lediglich mit einer hohen Wahrscheinlichkeit erreichen. Die beschriebenen Algorithmen besitzen quadratische bzw. lineare Laufzeiten. In beiden Algorithmen können Situationen auftreten, in denen noch nicht einsortierte Elemente ohne weitere Prüfung in einen neu erstellen Knoten einfügt werden müssen, da sonst die Bedingung der Minimalbelegung an m Elementen im Knoten nicht erfüllt werden kann. Zudem reicht die Minimierung der Gesamtgröße der erstellten AABBs als alleiniges Kriterium zu Bildung neuer Knoten nicht zwangsläufig aus. Das kann zu einer problematischen Verteilung von Datenobjekten in Blattknoten führen, wobei sich die AABBs einzelner Ebenen des Baums stark überlappen.

2.5.3 Die Indexstruktur R*-Tree

Der R*-Tree ist eine Variante, in der die Verteilung von indizierten Datenobjekten in Knoten verbessert wird (Beckmann et al., 1990). Im Fall zweidimensionaler Daten wird im R-Tree ausschließlich versucht, die Fläche neuer Knoten zu minimieren. Der R*-Tree nutzt hingegen mehrere Minimierungskriterien. Diese beziehen sich auf die Minimierung der Fläche und des Umfangs von Knotenbereichen. Außerdem wird die Überlappung zwischen Knoten minimiert. Dadurch steigt die Wahrscheinlichkeit, Auswertungen mit Hilfe der Datenstruktur in weniger Schritten ausführen zu können, was zu einer Effizienzsteigerung führt.

Um die genannten Minimierungen zu erreichen wird ein veränderter Algorithmus für die Auffindung des geeignetsten Knotens für neue Datenobjekte nötig. Da neue Knoten durch die Aufteilung bestehender Knoten bei deren Überlauf entstehen, ist auch an dieser Stelle der Algorithmus hinsichtlich der erweiterten Minimierungskriterien zu verändern. Abbildung 2.24 zeigt beispielhaft die unterschiedliche Aufteilung innerhalb eines R-Trees und eines R*-Trees.

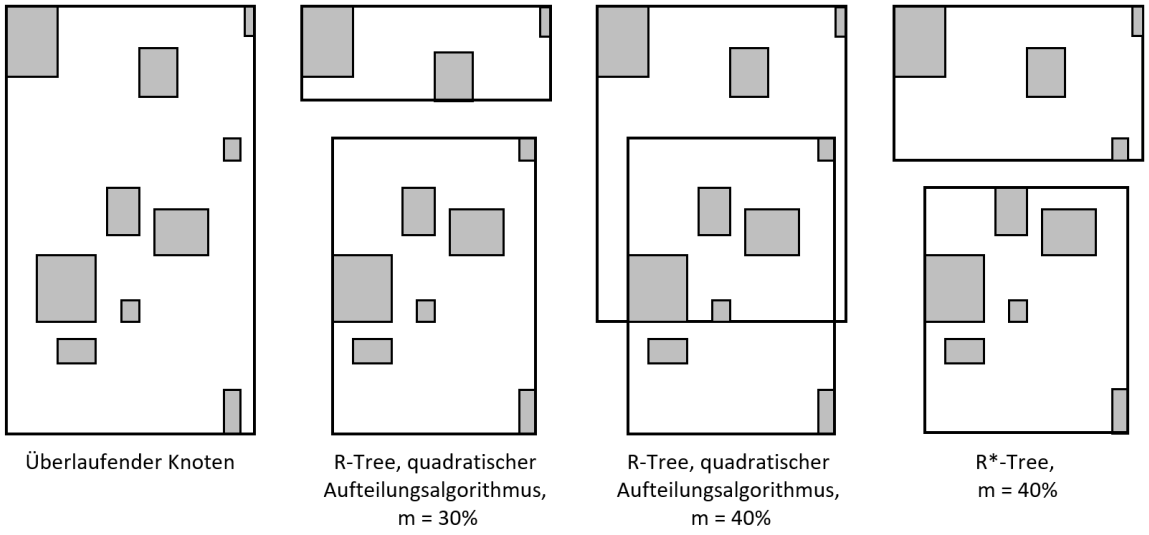


Abbildung 2.24: Unterschiedliche räumliche Aufteilung bei Knotenüberlauf innerhalb des R-Trees und des R*-Trees, nach (Beckmann et al., 1990)

Beide R-Tree Varianten sind nicht deterministisch in Bezug auf die letztendlich entstehende Baumstruktur. So erzeugt die gleiche Datengrundlage, die in unterschiedlichen Reihenfolgen indiziert wird, im Allgemeinen unterschiedliche Bäume. So haben die ersten Datenobjekte großen Einfluss darauf, wie die hierarchischen Knotenbereiche innerhalb des Baumes entstehen. Diese Bereiche sind nicht zwingend für später hinzukommende Datenobjekte optimal. Laufen diese über, erfolgt bei einer Neuaufteilung in zwei Knoten lediglich eine lokale Betrachtung der zu erzielenden Minimierungskriterien. Es hat sich gezeigt, dass durch ein zufälliges Löschen von 50% der bereits indizierten Objekte und einer darauffolgenden Neuindizierung das Suchverhalten des Baumes verbessert werden kann.

In (Beckmann et al., 1990, Kap. 5) werden der R-Tree und der R*-Tree ausführlich verglichen. Dies beinhaltet das Such- und Einfügeverhalten sowie die Ausnutzung des Speicherplatzes. Besonders bei den Suchoperationen erweist sich der R*-Tree als besonders performant, was auf die verbesserte räumliche Knotenaufteilung zurückzuführen ist.

Innerhalb BIM-basierten Methoden werden primär dreidimensionale Geometrierepräsentationen verwendet. In Kombination mit einer detailreichen Abbildung und einer hohen Anzahl an Bauteilen ergibt sich damit eine umfangreiche Datenbasis. Im Kontext einer Anfragesprache sollte die Ausführungszeit einzelner Anfragen möglichst kurz sein, um ein reaktionsfähiges Gesamtsystem zu verwirklichen. Auf Grund seiner hohen Selektionsleistung wird daher der R*-Tree als räumliche Indizierungsstruktur innerhalb der entwickelten Ablaufumgebung eingesetzt (siehe Kap. 5.2.3).

2.6 Fazit

Obwohl Gebäudemodelle und GIS-Daten jeweils räumliche Objekte und Konstrukte abbilden, können die Konzepte der Geoinformatik nicht ohne weitere Anpassungen in die modellbasierte Planung übertragen werden. Dies liegt hauptsächlich an den abweichenden Modellrepräsentationen und den unterschiedlichen Arbeitsprozessen in denen GIS- und BIM-Daten verwendet werden. Im Bauwesen ist naturgemäß die Errichtung neuer Bauwerke der fokussierte Anwendungsfall BIM-basierter Prozesse. Dahingegen stellt die langfristige, rechnergestützte Vorhaltung von bereits bestehenden Bauten eine Hauptaufgabe in der Geoinformatik dar.

Entscheidende Erweiterungen kann das Building Information Modeling jedoch erfahren, wenn Analysefunktionen aus der Geoinformatik übertragen werden können. Dies betrifft zum einen die Analyse zeitlicher Daten der baulichen Terminplanung. Außerdem können die räumlichen, im Speziellen die topologischen Funktionen der Geoinformationssysteme zu weitreichenden Analysen von BIM-Daten genutzt werden. Für die Überführung von GIS-Funktionen in die 3D-Gebäudemodellierung wurden in diesem Kapitel geeignete, räumliche Indizierungstechniken der Geoinformatik identifiziert.

Kapitel 3

Building Information Modeling

Die voranschreitende Digitalisierung und die Weiterentwicklungen in der Datenmodellierung und -verarbeitung veränderten in den letzten Jahrzehnten die Arbeitsweise im Bauwesen. Anstelle zweidimensionaler Zeichnungen, die in der konventionellen Planung einen der Hauptinformationsträger darstellten, werden Planungsdaten in computerinterpretierbaren Modellen vorgehalten. Die Weiterentwicklung der modellbasierten Planungsmethoden mündete in das *Building Information Modeling (BIM)*, einer neuartigen Prozessmethode im Bauwesen, in der rechnergestützte Gebäudemodelle die zentrale Rolle einnehmen.

Ein Gebäudemodell bündelt attributive und räumliche Bauteildaten, unter Bauteilen auftretende Beziehungen sowie zusätzliche planungsrelevante Informationen über Termine und Kosten. Eine derartige Repräsentation birgt für das Bauwesen ein hohes Maß an Qualitäts- und Effizienzsteigerung in sich. So liegt ein detailliertes Modell des Gebäudes vor, das mit computerbasierten Methoden vielfach genutzt, weiterverarbeitet und analysiert werden kann. Zudem werden Informationsbrüche und fehleranfällige manuelle Dateneingaben im Idealfall vermieden.

In diesem Kapitel werden die im BIM eingesetzten Modellierungstechniken besprochen. Nach der Erläuterung entsprechender Grundlagen und der Betrachtung der Modellierungssprache *EXPRESS* wird auf Speicherformate für Gebäudemodelle eingegangen. Darauf folgt die detaillierte Vorstellung der standardisierten und offenen *Industry Foundation Classes (IFC)*. Das Datenmodell ermöglicht eine herstellerneutrale Datenübertragung in BIM-basierten Prozessen und dient daher als primäre Datenquelle für das entwickelte Analysesystem. Neben der semantischen und geometrischen Modellierung ist in der Bauplanung die Vorhaltung und Beurteilung zeitlicher Abläufe nötig. Die Integration zeitlicher Informationen der Terminplanung in Gebäudemodelle wird daher ebenfalls in diesem Kapitel behandelt.

3.1 Grundlagen

Eastman et al. (2008, S. 16) definiert BIM als Prozessmethode im Bauwesen und als Modellierungstechnologie zur Erstellung, zur Kommunikation und zur Analyse von Gebäudemodellen. Zu unterscheiden ist somit zwischen der Methode und einem konkreten Modell. Letzteres wird in dieser Arbeit stets als Gebäudemodell bezeichnet. BIM basiert auf folgenden, grundlegenden Ansätzen:

- Bauteile werden als visualisierbare Objekte mit eindeutiger Identität vorgehalten und sind über Parameter veränderbar.
- Bauteile besitzen beschreibende Attribute und zeigen ein ihrer Semantik entsprechendes Verhalten.
- Ansichten und Pläne werden aus dem Gebäudemodell abgeleitet und befinden sich dadurch stets in einem konsistenten Zustand zum Ursprungsmodell.

Die sich daraus ergebende Modellierung in der Bauwerksplanung verspricht eine Reihe von positiven Effekten gegenüber dem konventionellen zeichnungsbasierten Vorgehen. Auf Basis von (Sacks et al., 2004, 2010; Eastman et al., 2008) lassen sich die Vorteile folgendermaßen zusammenfassen:

Einfache Erstellung von Design-Alternativen: Durch das intelligente Verhalten der parametrischen Bauelemente lassen sich Varianten im Vergleich zu CAD-Systemen mit weniger Arbeitsaufwand erstellen.

Einfache Erstellung von alternativen Bauabläufen: Durch die Einbeziehung von Konstruktionsmethoden für Bauteile, technologischen Abhängigkeiten und nötigen Ressourcen lässt sich die modellbasierte Bauablaufplanung teilautomatisieren und Varianten des Bauablaufs mit geringerem Aufwand generieren.

Kollaborativer Entwurf und Planung: Hierdurch können Projektbeteiligte synchron oder asynchron an einem Modell arbeiten. Die Kollaboration kann innerhalb eines Gewerks oder gewerkeübergreifend stattfinden.

Einsatz von Client-Server-Technologie: BIM-Server dienen zur zentralen Verwaltung von Gebäudemodellen und bieten einen erweiterten Funktionsumfang gegenüber Desktop-Systemen. Dazu zählen Mehrbenutzerbetrieb, die Versionierung von Gebäudemodellen und eine höhere Datensicherheit bei Hardwareausfällen.

Abschätzbarkeit der Gebäude-Performance: Es können Aussagen beispielsweise über den erwartenden Engergieverbrauch und entsprechende Unterhaltskosten getroffen wer-

den. Auch eine Verifizierung des geplanten Gebäudes gegenüber verbindlichen Raumprogrammen und Baunormen ist möglich.

Redundanzvermeidung und Integritätssicherung: Dieselben Daten werden in Plänen beispielsweise in Übersichts- und Detailzeichnungen wiederholt dargestellt, wodurch sich Inkonsistenzen ergeben können. Im Gegensatz dazu agiert ein Gebäudemodell als zentrale Datenbasis für Entscheidungsprozesse. Durch die redundanzfreie Modellierung und zusätzliche Funktionalitäten wie *Clash Detection* kann die Modellintegrität gewährleistet werden (Gijezen et al., 2010).

Die genannten Vorteile kommen auf Basis der vorhandenen Konzepte und Technologien im Building Information Modeling jedoch nur teilweise zum Tragen und es ergeben sich zudem neue Fragestellungen. Werden beispielsweise Modellvarianten bezüglich Design und Bauablauf erstellt, können die Unterschiede zwischen den einzelnen Modellen nicht computergestützt und in passender Form ermittelt werden. Auch im Rahmen einer kollaborativen Arbeitsweise und beim Einsatz von Client-Server-Technologie stellt sich die Frage, wie sich Modellveränderungen erkennen lassen und wie Teilmodelle automatisch extrahiert und zusammengeführt werden können. Des Weiteren erfordert die Verifikation gegenüber Raumprogrammen und Normen trotz BIM-basierter Planung eine größtenteils manuelle Überprüfung. Ferner ist mit Clash Detection, dem Auffinden von geometrischen Überlagerungen zwischen Bauteilen, nur ein Teil der möglichen räumlich-semantischen und raum-zeitlich-semantischen Inkonsistenzen innerhalb eines Gebäudemodells identifizierbar. Die genannten Defizite sind Resultat eines fehlenden, domänenspezifischen Konzepts zur formalen, computergestützten Analyse von Gebäudemodellen.

Im nächsten Abschnitt wird auf die Datenmodellierung im Bauwesen, speziell im Kontext der Modellierungssprache EXPRESS und deren Speicherformate eingegangen.

3.2 Datenmodellierung im Bauwesen

Innovative Bauvorhaben sollen wirtschaftlich, innerhalb eng bemessener Zeiträume und in hoher Qualität realisiert werden. In Anbetracht des Unikatcharakters eines Gebäudes und variierender Beteiligten unterscheiden sich Bauprojekte stark voneinander. Dies schließt vertragliche Gegebenheiten, Verantwortungen, Projektziele, Budgets und Realisierungszeiträume mit ein (Huhnt, 2004). Eine langfristige Verkettung von Softwareprodukten auf Basis proprietärer Schnittstellen und Individualanpassungen ist auf Grund dieser wechselnden Projektbedingungen nicht sinnvoll.

Um Planungsdaten mit hohem inhaltlichem Niveau zwischen der Vielzahl an Projektbeteiligten und den entsprechenden BIM-Werkzeugen effizient und herstellerneutral austauschen zu können, bedarf es offener Datenmodelle und entsprechender Formate. Das umfassendste, domänenübergreifende und offene Produktmodell im BIM-Bereich sind die Industry Foundation Classes. Zur Definition dieses Modells werden Methoden aus bereits vorhandenen ISO-Standards eingesetzt, die im Anschluss erläutert werden.

Bereits seit 1984 wird seitens der ISO an einem allgemeinen Standard zur Repräsentation und zum Austausch von Produktmodellen gearbeitet (Pratt, 2001). Aus dieser Bestrebung entstand der *Standard for the Exchange of Product model data (STEP)*, der sich aus mehreren Teilbereichen zusammensetzt. Diese beinhalten für sich ebenfalls valide Standards (ISO, 1994). Zu unterscheiden sind Spezifikationen für konkrete Implementierungen (*Application Protocols*) und allgemeingültige, wiederverwendbare Definitionen (*Integrated Resources*).

Auf Basis von Application Protocols werden in STEP spezialisierte Schemata für die Produktmodellierung, unter anderem in Bereichen des Maschinenbaus und der Elektrotechnik vorgehalten. Die Definition eines umfassenden, generellen Schemas für Gebäudemodelle stellt eine komplexe Modellierungsaufgabe dar. Nach einem anfänglichen Versuch die Modellierung innerhalb von STEP zu verwirklichen, wurde diesbezüglich ein eigenständiges Projekt ins Leben gerufen (Laakso & Kiviniemi, 2012, Kap. 3).

So ist seit dem Jahre 1994 mit der IFC 1.0 ein generelles, offenes Produktmodell für die Bauindustrie verfügbar. Zum Zeitpunkt der Drucklegung dieser Arbeit ist Version 4 der aktuelle Stand des Produktmodells. Auf Grund von Anwendungsüberschneidungen verwendet das IFC-Datenmodell, zur Redundanzvermeidung, Teile der Geometriedefinition aus dem STEP-Standard. Außerdem wird die in STEP enthaltene Modellierungssprache EXPRESS zur Schemaspezifikation der IFC eingesetzt (Gu & Chan, 1995, S. 3). Im Anschluss werden grundlegende Konzepte einer EXPRESS-basierten Modellierung veranschaulicht.

3.2.1 EXPRESS-basierte Datenmodellierung

EXPRESS ist eine deskriptive Modellierungssprache, die es erlaubt, Entitäten innerhalb einer Vererbungshierarchie zu beschreiben. Jede Entitätsklasse wird durch Attribute, Relationen zu anderen Klassen und *Einschränkungen* (engl. *constraints*) beschrieben. Alle definierten Klassen bilden in ihrer Gesamtheit das eigentliche Schema des Produktmodells, wie beispielsweise das der IFC. Auf Basis von (Schenck & Wilson, 1994, S. 17) und (Fowler, 1995, Anh. B) werden im Folgenden die Grundelemente von EXPRESS vorgestellt.

Schema: Ein Produktmodell wird auf Basis eines oder mehrerer Schemata festgelegt, die aus Entitäten und Typen bestehen.

Simple Type: Dies umfasst einfache grundlegende Typen wie Zeichenketten, natürliche Zahlen und Fließkommazahlen.

User-defined Type: Auf Basis der einfachen Typen lassen sich benutzerdefinierte, eingeschränkte Typen definieren, die bereits über ein geringes Maß an Semantik verfügen.

Select Type: Dieser Typ erlaubt die Erstellung einer Superklasse für unterschiedliche Typen und Entitäten. Die Select-Superklasse ist dabei nicht Teil der Vererbungshierarchie und agiert als Enumeration auf Typebene.

Container: Dies sind Typen zur Repräsentation geordneter und nicht-geordneter *Container* wie ARRAY, BAG, SET und LIST. Unterschieden wird außerdem ob ein Objekt mehrfach in der Ansammlung enthalten sein kann und ob die Anzahl an Elementen innerhalb des Containers fest oder variabel ist.

Entity: Die Entität stellt das **zentrale Modellierungskonstrukt** in EXPRESS dar und dient zur Repräsentation komplexer Objekte.

Attribute: Die Eigenschaften von Entitäten werden durch ihre typisierten Attribute ausgedrückt.

Constraint: Eine Einschränkung erlaubt die Definition von Regeln für Attributwerte und Entitäten.

Eine Entität als Grundelement der objektorientierten Modellierung ist in EXPRESS meist Teil einer Vererbungshierarchie. Dadurch lassen sich Spezialisierungen ableiten und Redundanzen in der Modellierung vermeiden. Eine Eltern-Entität vererbt alle Attribute und Einschränkungen an die jeweilige Kind-Entität. Einschränkungen erlauben es, *Regeln* (engl. *rules*) für Attribute aufzustellen. Somit lassen sich beispielsweise eindeutige Objektidentifizierer (*unique rule*), Bedingungen für Attributwerte (*where rule*) und Attributpaare, die eine bidirektionale Verbindung zwischen zwei Entitätsklassen darstellen (*inverse rule*), definieren. Ein Attribut kann außerdem als optional oder *funktional abhängig* deklariert werden. Im letzten Fall wird das Attribut mit Hilfe weiterer Attribute und einer Berechnungsvorschrift bestimmt. EXPRESS umfasst eine Vielzahl weiterer Konzepte. Diesbezüglich sei auf (TC184, 2004) und (Schenck & Wilson, 1994, Kap. 10-15) verwiesen.

Zur Verdeutlichung der Konzepte der EXPRESS-basierten Modellierung wird die Definition einer Entität betrachtet. Ein IfcTask stellt innerhalb des Produktmodells ein in sich abgeschlossenes Arbeitspaket dar. Auflistung 3.1 zeigt den entsprechenden Ausschnitt des IFC4-Schemas.

IfcTask erbt von IfcProcess und fügt dieser Definition sechs weitere Attribute hinzu. In der EXPRESS-Sprache können Attribute sowohl benutzerdefinierte Typen (Z. 4) und einfache Typen (Z. 5) aufnehmen, sowie Entitäten referenzieren (Z. 7). Die Definition des benutzerdefinier-

ten Typs IfcLabel erfolgt in Zeile 14. In den nachfolgenden Zeilen wird der im *PredefinedType*-Attribut genutzte Enumerationstyp festgelegt. Die IfcTask-Definition enthält des Weiteren die zwei Zwangsbedingungen `HasName` und `CorrectPredefinedType` (Z. 10 u. 11). In der ersten Bedingung wird sichergestellt, dass das von IfcRoot geerbte *Name*-Attribut belegt ist. Die zweite Zwangsbedingung prüft die korrekte Belegung des *PredefinedType*-Attributs.

```
1  ENTITY IfcTask
2   SUBTYPE OF (IfcProcess);
3    Status : OPTIONAL IfcLabel;
4    WorkMethod : OPTIONAL IfcLabel;
5    IsMilestone : BOOLEAN;
6    Priority : OPTIONAL INTEGER;
7    TaskTime : OPTIONAL IfcTaskTime;
8    PredefinedType : OPTIONAL IfcTaskTypeEnum;
9   WHERE
10   HasName : EXISTS(SELF\IfcRoot.Name);
11   CorrectPredefinedType : NOT(EXISTS(PredefinedType)) OR (PredefinedType <>
     IfcTaskTypeEnum.USERDEFINED) OR ((PredefinedType = IfcTaskTypeEnum.
     USERDEFINED) AND EXISTS(SELF\IfcObject.ObjectType));
12 END_ENTITY;
13
14 TYPE IfcLabel = STRING(255); END_TYPE;
15
16 TYPE IfcTaskTypeEnum = ENUMERATION OF (ATTENDANCE, CONSTRUCTION, DEMOLITION,
   DISMANTLE, DISPOSAL, INSTALLATION, LOGISTIC, MAINTENANCE, MOVE, ...);
17 END_TYPE;
```

Auflistung 3.1: Ausschnitt aus dem IFC 4 Schema (buildingSMART International Ltd, 2013c)

Mittels EXPRESS definierte Datenmodelle werden üblicherweise in Dateien mit der Endung `exp` abgelegt. Tabelle 3.1 gibt einen Überblick über den Umfang des IFC4-Schemas im Vergleich zu den früheren Versionen ab 2X. Die Daten wurden aus den entsprechenden Schemadateien ermittelt.

Schema	IFC2X	IFC2X2	IFC2X3	IFC4
Entities	370	629	653	766
User-defined Types	89	110	117	126
Enumerations	116	161	164	206
Select Types	23	42	46	59

Tabelle 3.1: Umfang des IFC4-EXPRESS-Schemas im Vergleich zu früheren IFC-Versionen ab 2X

Neben der vorgestellten textuellen Modellierungssprache existiert mit *EXPRESS-G* eine graphische Notation. Abbildung 3.1 zeigt beispielhaft die Definition der IfcTask-Entität als EXPRESS-G-Diagramm. Neben der Ableitung von IfcProcess und den hinzugekommenen

Attributen ist in diesem Diagramm die Verwendung von IfcTask in einer Abfolgebeziehung (IfcRelSequence) ersichtlich. Für eine Erläuterung der graphischen EXPRESS-G Notation sei auf (Stanford University, 2008) verwiesen.

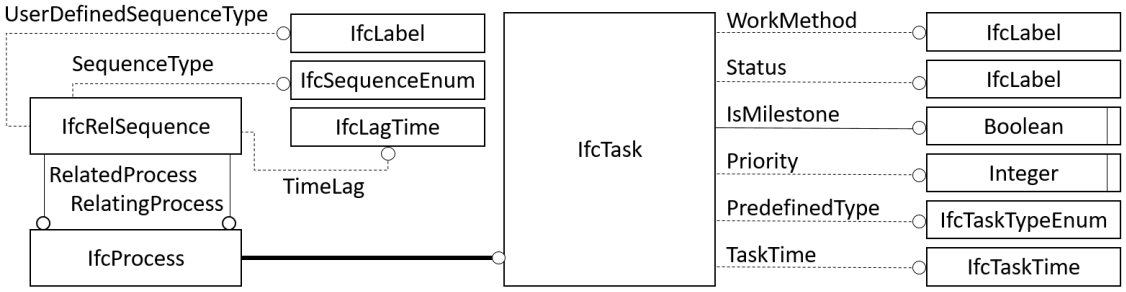


Abbildung 3.1: Visualisierung der IfcTask-Entität, nach (buildingSMART International Ltd, 2013b), EXPRESS-G-Diagramm

3.2.2 Speicherformate für Modellinstanzen

Zur Repräsentation von Modellinstanzen eines EXPRESS-Schemas wird im *Part 21* des STEP-Standards ein textbasiertes Austauschformat spezifiziert (ISO, 2002a). Eine entsprechende Instanzdatei, auch als *STEP Physical File* oder *P21 File* bezeichnet, teilt sich in Header- und Datenbereich auf. Der Header besteht aus Meta-Informationen wie Angaben zum Autor, Zeit der Erstellung, verwendetes Bearbeitungsprogramm und EXPRESS-Schema.

In der Datensektion sind Instanzen aus dem verwendeten Schema zeilenweise aufgelistet. Dabei wird jeder Entität eine im Umfang der Datei eindeutige Objektbezeichnung zugewiesen. Diese wird aus dem #-Zeichen und einem fortlaufenden Integerwert gebildet und zur Objektidentifizierung in Referenzbeziehungen genutzt. Die Objektbezeichnung ist selber kein beschreibendes Entitätsattribut, sondern wird dynamisch beim Export des Modell erstellt. Zur eindeutigen Referenzierung von Instanzen über die Grenzen einer einzelnen Datei hinaus dient hingegen ein spezielles Attribut (siehe IfcRoot-Klasse in Abschn. 3.3.2).

Auflistung 3.2 zeigt zur Illustration eine IfcTask-Instanz innerhalb einer P21-Datei. Die Entitätsrepräsentation beginnt mit der Objektbezeichnung `#3123`. Danach folgt der Klassenname, hier `IFCTASK`. In den anschließenden Klammern werden, durch Komma getrennt, alle Attributwerte angegeben. Zeichenketten werden dabei in Hochkomma und Enumerationswerte in Punkte eingefasst. Werden Entitätsinstanzen in Attributen referenziert, erfolgt die Nennung der entsprechenden eindeutigen Objektbezeichnung (`#29`, siehe zweites Attribut). Optionale Attribute können mit Null-Werten belegt werden, die durch `$`-Zeichen repräsentiert werden. Bei der Verwendung benutzerdefinierter Typen in Attributen

muss ein entsprechender Typkonstruktor wie beispielsweise `IFCBOOLEAN()` genutzt werden.

```
#3123= IFCTASK('01EcURliX1lRFLJ0rXw_JP',#29,'Task-004',$,$,12,'Started','
concreting',IFCBOOLEAN(.T.),3);
```

Auflistung 3.2: Eine IfcTask-Instanz innerhalb einer ISO10303-21-konformen Datei

Neben der Vorhaltung gemäß der *ISO10303-21* kann eine IFC-Instanzdatei auch in einer XML-basierten Repräsentation vorliegen. Schemata für derartige Datenmodelle werden in der *XML Schema Definition Language (XSD)* aufgestellt (Fallside & Walmsley, 2004). Für die Ableitung eines XSD-Schemas aus einem EXPRESS-Schema existiert ein standardisiertes, in der *ISO10303-28* beschriebenes Verfahren (ISO, 2007a). Dieses wird auch zur Ableitung einer XML-basierten IFC-Modellierung verwendet (Liebich, 2001). Das gewonnene, *markupbasierte* Schema wurde in früheren IFC-Versionen in zwei Dateien aufgeteilt. Mit Version 4 liegt das komplette Schema erstmals in einer Datei vor, wodurch die Validierung von Instanzdateien erleichtert wird (buildingSMART International Ltd, 2013a). Aufzählung 3.3 zeigt die XSD-basierte Definition der IfcTask-Entität.

```
<xs:element name="IfcTask" type="ifc:IfcTask" substitutionGroup="ifc:
IfcProcess" nillable="true"/>
<xs:complexType name="IfcTask">
  <xs:complexContent>
    <xs:extension base="ifc:IfcProcess">
      <xs:sequence>
        <xs:element name="TaskTime" type="ifc:IfcTaskTime" nillable="true"
        minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Status" type="ifc:IfcLabel" use="optional"/>
      <xs:attribute name="WorkMethod" type="ifc:IfcLabel" use="optional"/>
      <xs:attribute name="IsMilestone" type="xs:boolean" use="optional"/>
      <xs:attribute name="Priority" type="xs:long" use="optional"/>
      <xs:attribute name="PredefinedType" type="ifc:IfcTaskTypeEnum" use="
      optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Auflistung 3.3: Die IfcTask-Definition im XSD-Schema der IFC4, aus (buildingSMART International Ltd, 2013a)

3.3 Konzepte der Industry Foundation Classes

Der IFC-Standard wird von der internationalen Non-Profit-Organisation *buildingSMART* spezifiziert. Diese besteht aus einer Vielzahl an Mitgliedern aus Wirtschaft, Forschung und Ver-

waltung. Die Organisation versucht den Übergang zu modernen Verfahren in der Bauindustrie zu erleichtern. Dazu veröffentlicht sie offene, internationale Standards im BIM-Bereich (BuildingSMART, 2015). Im Folgenden werden grundlegende Konzepte der IFC-basierten Gebäudemodellierung vorgestellt.

3.3.1 Schichten der IFC

Das IFC-Produktmodell ist konzeptionell in die vier Modellierungschichten *Domain Layer*, *Interop Layer*, *Core Layer* und *Resource Layer* aufgeteilt. Diese sind inklusive der jeweils enthaltenen Subschemata in Abbildung 3.2 dargestellt. Im Folgenden wird jede Ebene inklusive eines exemplarischen Subschemas erläutert.

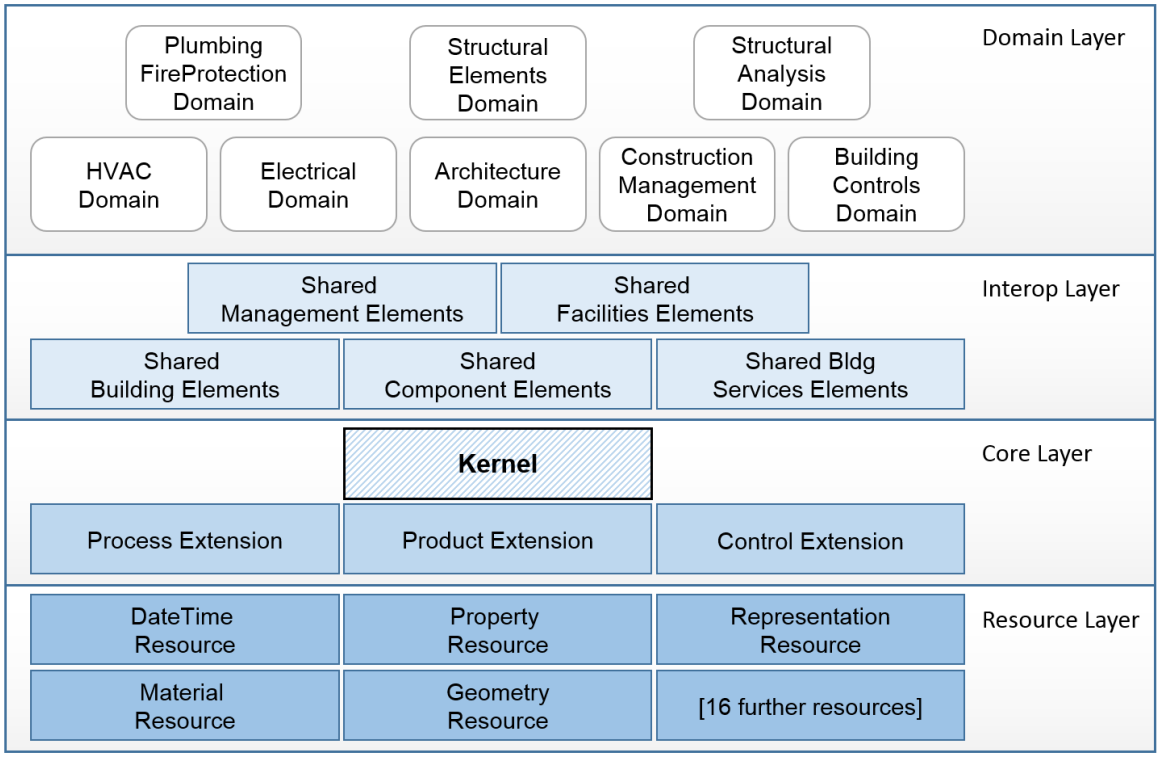


Abbildung 3.2: Layerstruktur des IFC-Produktmodells, nach (Liebich et al., 2013)

Der **Domain Layer** beinhaltet Schemata, die Entitätsdefinitionen von Produkten, Prozessen und Ressourcen für einen domäneninternen Datenaustausch bereitstellen. Das Subschema *Stuctural Elements Domain* definiert strukturelle Bauwerksteile wie Fundamente und Pfähle sowie Bewehrungen für diese.

Der **Interop Layer** veröffentlicht Schemata für die Definition von Produkten, Prozessen und Ressourcen für einen domänenübergreifenden Datenaustausch. Das Subschema *Shared Building Elements* beinhaltet spezialisierte Komponenten für die Architekturplanung. Beispiele hierfür sind Wände, Stützen und Träger.

Der **Core Layer** enthält das Kernschema (engl. kernel) der IFC und drei Erweiterungsschemata für Kontroll-, Produkt- und Prozess-Entitäten. Im Kernel sind die generellsten Entitätsdefinitionen wie beispielsweise IfcRoot und IfcObject hinterlegt. Von diesen Entitäten werden innerhalb höherliegender Schichten eine Vielzahl an Subklassen abgeleitet.

Der **Resource Layer** beinhaltet vielfältige, wiederverwendbare Schemadefinitionen. Entitäten dieser Schicht besitzen innerhalb des Produktmodells keine eindeutige Identität und können nicht ohne beherbergendes Objekt genutzt werden. Das Subschema *DateTime Resource* stellt zum Beispiel Entitäten und Typen zur Vorhaltung von Kalender- und Zeitdaten bereit.

Das Layerkonzept der IFC dient zur Gruppierung verwandter Klassen und zur Festlegung von Referenzierungsregeln zwischen einzelnen Ebenen. So dürfen Entitäten keine Typ- und Entitäts-Definitionen in ihren Attributen nutzen, die aus übergeordneten Ebenen stammen.

3.3.2 Identitätsbehaftete Modellierung

IfcRoot ist die abstrakte Superklasse einer jeden selbständigen IFC-Entität (siehe Auflistung 3.4). Sie verleiht Instanzen erbender Klassen eine eindeutige Identität. Dadurch ist es diesen Instanzen erlaubt, eigenständig im Objektmodell aufzutreten, ohne dabei in andere Entitäten eingebettet werden zu müssen. Die eindeutige Objektidentität wird durch das Attribut *GlobalId* realisiert, das eine – praktisch gesehen – weltweit eindeutige Objektbezeichnung aufnimmt. IfcRoot gibt drei weitere Attribute an erbende Entitäten weiter. Durch das *OwnerHistory*-Attribut kann jede IfcRoot-Instanz den letzten Bearbeiter vorhalten. Die zwei optionalen Attribute *Name* und *Description* dienen zur weiteren Beschreibung von Entitäten.

```
ENTITY IfcRoot
 ABSTRACT SUPERTYPE OF (ONEOF (IfcObjectDefinition, IfcPropertyDefinition,
 IfcRelationship));
  GlobalId : IfcGloballyUniqueId;
  OwnerHistory : OPTIONAL IfcOwnerHistory;
  Name : OPTIONAL IfcLabel;
  Description : OPTIONAL IfcText;
 UNIQUE
  UR1 : GlobalId;
END_ENTITY;
```

Auflistung 3.4: Die IfcRoot-Definition aus dem IFC EXPRESS-Schema Version 4 (buildingSMART International Ltd, 2013c)

3.3.3 Repräsentation von Bauteilen

Innerhalb des Interop Layers des IFC-Produktmodells wird das Subschema Shared Building Elements definiert. Es enthält Entitäten für die Modellierung von Bauwerkskomponenten wie Wände, Stützen und Bodenplatten. Als Hauptelemente der architektonischen Planung prägen diese Komponenten das Erscheinungsbild des Bauwerks und bestimmen dessen Funktionalität. Die angewendete Modellierung basiert auf einer tiefen Vererbungshierarchie, die am Beispiel von Wänden in Abbildung 3.3 dargestellt ist.

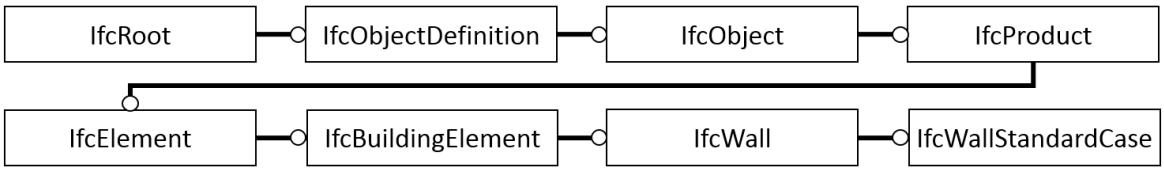


Abbildung 3.3: Vererbungshierarchie für die IfcWallStandardCase-Entität ausgehend von IfcRoot, EXPRESS-G-Diagramm

Nach der bereits besprochenen IfcRoot-Klasse folgt in der nächsten Stufe der Vererbungshierarchie IfcObjectDefinition. Instanzen von IfcObjectDefinition können über explizite Beziehungsrepräsentationen, sogenannten *Objectified Relationship*s mit anderen Entitäten verknüpft werden. Das Konzept wird im anschließenden Abschnitt 3.3.4 näher erläutert.

Das nachgelagerte IfcObject kann in einer Typzuweisung verwendet werden. Dadurch gehen die Eigenschaften eines Typs (IfcTypeObject) auf die Bauwerkskomponenten über. Es können sowohl Geometrie- als auch Attributdefinitionen übertragen werden. Außerdem ist die Verknüpfung eines IfcObjects zu einer Merkmalsliste (IfcPropertySetDefinition) möglich (siehe Abschn. 3.3.6).

Das nachfolgende IfcProduct dient zur Modellierung räumlicher Objekte. Über das Attribut *ObjectPlacement* wird die Lage und über das Attribute *Representation* die Form eines Objekts bestimmt. Die Lagebeschreibung kann dabei absolut im Weltkoordinatensystem, relativ zu anderen Entitäten oder bedingt durch Layout-Elemente wie einem Gitter geschehen. Zur Definition der räumlichen Form können eine oder mehrere Geometrierepräsentationen (IfcGeometricRepresentationItem) verwendet werden (siehe Abschn. 3.3.7).

Das anschließende IfcElement agiert als Generalisierung für Bauelement. IfcElements können in einer Vielzahl objektivierter Beziehungen genutzt werden und definieren daher spezielle *inverse Attribute* (siehe Abschn. 3.3.5).

Nach der weiteren Spezialisierung in IfcBuildingElement, das die Modellierung von Belägen auf Bauteilen ermöglicht, wird IfcWall und IfcWallStandardCase in der Vererbungshierarchie der Shared Building Elements erreicht. IfcWallStandardCase repräsentiert dabei den Standardfall

vertikal extrudierter Wände. Im Anhang ist die gesamte Vererbungshierarchie inklusive aller Attributdefinitionen für IfcWallStandardCase dargestellt (siehe Abschn. A.3).

Die intensive Nutzung von Vererbungsbeziehungen innerhalb der IFC-Schemas hat Auswirkungen auf die Analyse entsprechender Datenbestände. Es ergibt sich die Möglichkeit, Fragestellungen vermehrt auf Basis von Typuntersuchungen zu beantworten. Beispielsweise kann die Selektion aller physischer Bauteile eines Modells durch die Typabfrage gegenüber IfcBuildingElement erfolgen, die Selektion aller räumlichen Strukturen wie Stockwerke und Räume durch Abfrage gegenüber IfcSpatialStructure. Daraus folgt, dass in einer BIM-Anfragesprache eine Typselektion mit Berücksichtigung der Vererbungshierarchie benötigt wird. Dies mündet in die Entwicklung des TypeFilter-Operators in Kap. 6.5.2.

3.3.4 Objektivierte Beziehungen

Die Elemente eines Gebäudemodells beeinflussen sich in vielfältiger Weise gegenseitig. Dies kann in einer computerinterpretierbaren Repräsentation durch eine Beziehungsmodellierung ausgedrückt werden. Prinzipiell ist die Vorhaltung einer Reihe von Beziehungstypen innerhalb eines Gebäudemodells von Interesse. Die IFC verwirklicht diesbezüglich ein spezielles Modellierungskonzept. Hierbei werden Beziehungen durch explizite, identitätsbehaftete Objekte abgebildet, die im Englischen als Objectified Relationships bezeichnet werden. Das Konzept wird in Abbildung 3.4 verdeutlicht.

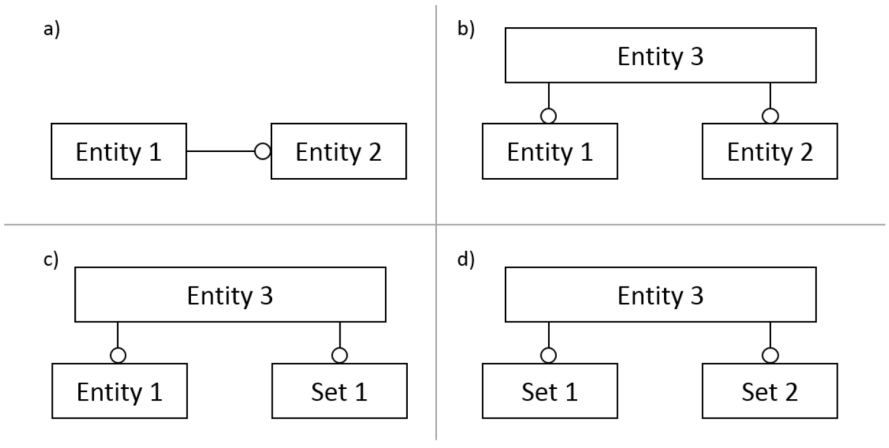


Abbildung 3.4: Eine direkte Referenzierung (a) und Varianten objektivierter Beziehungen (b-d), EXPRESS-G-Diagramme

Im Gegensatz zu einem direkten Verweis (a) wird das Relationsobjekt `Entity 3` genutzt, das zwischen den beiden weiteren Entitäten vermittelt (b). In der IFC Version 4 existieren 42

derartiger instanziierbaren Relationsentitäten. Mit 26 Ausprägungen beschreibt der überwiegende Teil dabei 1:n-Beziehungen (c). Beziehungen mit n:m-Charakter, wie in (d) dargestellt, existieren in der IFC derzeit nicht.

Für die Benennung der verknüpfenden Attribute eines Beziehungsobjekts gilt folgende Konvention: Auf das führende Objekt verweist ein mit `Relating` beginnendes Attribut. Ein oder mehrere zu verknüpfende Objekte werden über ein mit `Related` beginnendes Attribut referenziert. Abbildung 3.5 zeigt die Entitäten IfcRelAssigns, IfcRelAssociates, IfcRelConnects, IfcRelDeclares, IfcRelDecomposes und IfcRelDefines. Diese bilden nach IfcRelationship die zweite Ebene in der Vererbungshierarchie zur Repräsentation objektivierter Beziehungen und werden im Folgenden näher beschrieben.

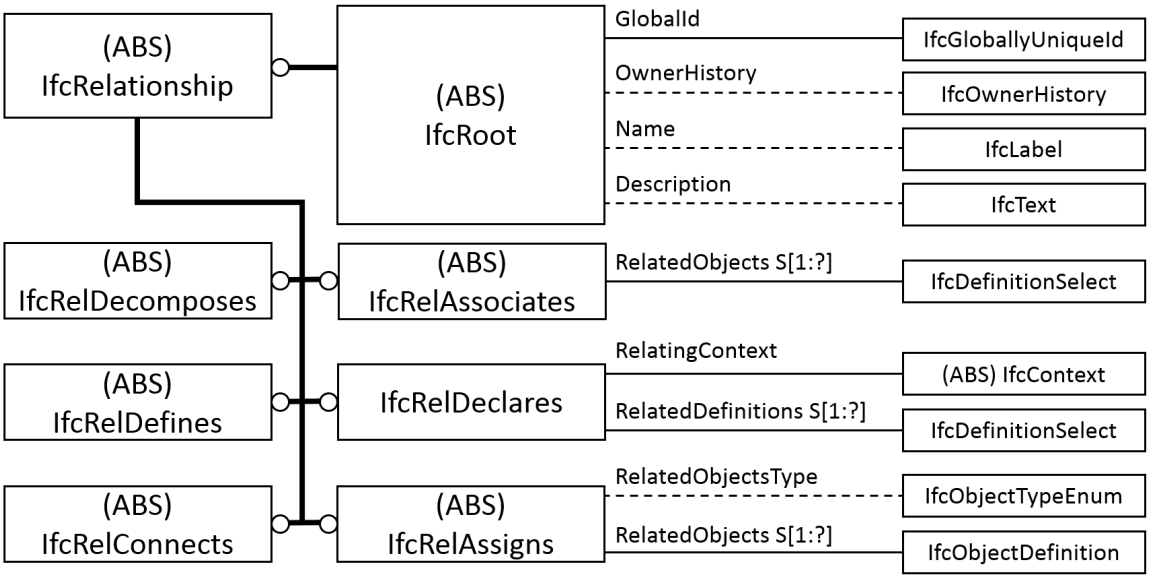


Abbildung 3.5: Erste und zweite Ebene der Vererbungshierarchie objektivierter Beziehungen in der IFC Version 4, EXPRESS-G-Diagramm

IfcRelAssigns: Eine bidirektionale Assoziation zwischen zwei Entitäten oder zwischen einer Entität und einer Entitätsgruppe. Die Semantik der führenden Entität wird in Spezialisierungen wie IfcRelAssignsToProcess und IfcRelAssignsToProduct definiert.

IfcRelAssociates: Eine Assoziation zu einer externen Ressource wie beispielsweise einer Bibliothek.

IfcRelConnects: Eine verbindungsbasierte Beziehung zwischen Entitäten. Die Semantik der Verbindung wird in Subklassen bestimmt.

IfcRelDeclares: Der Beziehungstyp verknüpft Entitäts- und Merkmalsdefinitionen mit einem Projekt (IfcProject) oder mit einer Projektbibliothek (IfcProjectLibrary).

IfcRelDefines: Subklassen dieses Typs werden genutzt, um Entitäten mit Typdefinitionen und Definitionen von Merkmalslisten zu assoziieren.

IfcRelDecomposes: Mit diesem Beziehungstyp kann eine Teil-Ganzes-Hierarchie realisiert werden.

Als Beispiel für die Definition eines Beziehungsobjekts zeigt Auflistung 3.5 einen Schemaausschnitt bezüglich IfcRelConnectsElements. Als Spezialisierung von IfcRelConnects repräsentiert diese Entität logische und reale Verbindungen zwischen Elementen. Über das Attribut *ConnectionGeometry* kann das Beziehungsobjekt eine Geometrie zur räumlichen Repräsentation der Verbindung aufnehmen.

```
ENTITY IfcRelConnectsElements
  SUPERTYPE OF(ONEOF(IfcRelConnectsPathElements,
  IfcRelConnectsWithRealizingElements))
  SUBTYPE OF IfcRelConnects;
    ConnectionGeometry  : OPTIONAL IfcConnectionGeometry;
    RelatingElement : IfcElement;
    RelatedElement  : IfcElement;
  WHERE
    NoSelfReference : RelatingElement :<>: RelatedElement;
END_ENTITY;
```

Auflistung 3.5: Beispiel einer Beziehungsobjekt-Definition, Auszug aus dem IFC4-Schema (buildingSMART International Ltd, 2013c).

Ein Beispiel aus dem Bereich der Haustechnik demonstriert die Anwendung objektivierter Relationsobjekte. Drei unterschiedliche Beziehungsarten werden dabei anhand des Rohrleitungssystems einer Klimaanlage behandelt. Innerhalb des IFC-Modells dient die IfcPipeSegment-Entität zur Repräsentation eines einzelnen Rohrsegments. Folgende Beziehungen können sich unter anderem für ein derartiges Segment ergeben:

- Es befindet sich innerhalb einer räumlichen Struktur wie zum Beispiel einem Stockwerk oder einem Raum.
- Die IfcPipeSegment-Instanz ergibt mit anderen Elementen das Rohrleitungssystem. Das Rohrstück ist demnach Teil eines übergeordneten Ganzen.
- Die IfcPipeSegment-Instanz überschneidet sich mit einem anderem Rohrstück, da auf die detailliertere Modellierung mit einem Verbindungsstück auf Grund des Planungsstandes verzichtet wurde.

Zur Modellierung der beschriebenen Eigenschaften des Rohrleitungssystems können folgende spezialisierte Beziehungsobjekte eingesetzt werden:

- Um das IfcPipeSegment einer räumlichen Struktur zuzuordnen dient das Beziehungsobjekt IfcRelContainedInSpatialStructure. Dieses leitet sich direkt von IfcRelConnects ab.
- Zur Beschreibung von spezialisierten Teil-Ganzes-Beziehungen wird IfcRelAggregates, Subkasse von IfcRelDecomposes, genutzt. Dabei ergeben die geometrischen Repräsentationen der Teile die räumliche Form des Ganzen.
- Sich schneidende Rohrstücke können über IfcRelInterferesElements miteinander assoziiert werden. IfcRelInterferesElements ist eine direkte Ableitung von IfcRelConnects.

Objektivierte Relationen ermöglichen eine detaillierte und dynamische Beziehungsbeschreibung auf Instanzebene. Die Auswertung dieser Beziehungen zwischen Entitäten kann als Ansatzpunkt für Analysen von Gebäudemodellen dienen. Daher muss ein BIM-Anfragesystem eine direkte Dereferenzierung von Entitätsattributen unterstützen (siehe Dereferencer-Operator in Kap. 6.5.2).

3.3.5 Inverse Attribute

Das Konzept der Objectified Relationsships erlaubt eine dynamische Erstellung von Relationen im Produktmodell. Dabei verweist ein Beziehungsobjekts auf die zu assoziierenden Entitäten. Als problematisch erweist sich unter Umständen die vom Beziehungsobjekt ausgehende Navigation. Um eine bidirektionale Navigation zu realisieren, nutzt das IFC-Datenmodell das EXPRESS-basierte Konzept inverser Attribute.

Abbildung 3.6 a) zeigt das Beziehungsobjekt *Entity 3* mit seinen direkten Attributen *Attribute1* und *Attribute2*. Durch die inversen Attribute *IAttribute1* und *IAttribute2* kann von *Entity 1* zu *Entity 3* und von *Entity 2* zu *Entity 3* navigiert werden. In Abbildung 3.6 b) wird das Zusammenspiel von IfcRelConnectsElements- und IfcElements-Instanzen über ihre direkten und inversen Attribute dargestellt.

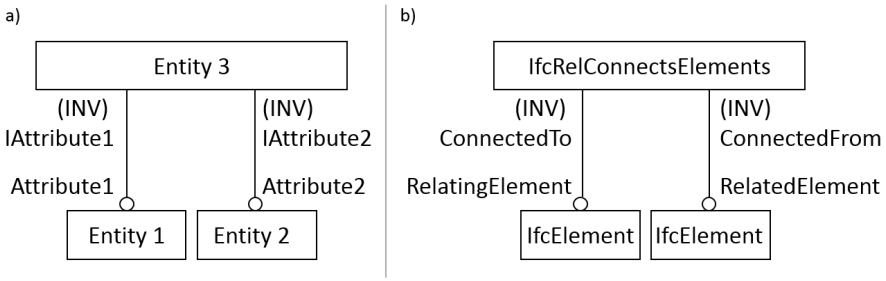


Abbildung 3.6: Veranschaulichung des EXPRESS-basierten Konzepts inverser Attribute, a) abstraktes Beispiel, b) Beispiel zu IfcRelConnectsElements, EXPRESS-G-Diagramme

Auflistung 3.6 zeigt Teile der Definition von IfcElement. Jedes inverse Attribut wird mit dem entsprechenden standardmäßigen Attribut des Beziehungsobjekts explizit verbunden. Insgesamt beinhaltet IfcElement 15 inverse Attribute, die zu optional vorhandenen Relationsobjekten verweisen können. Im Anhang werden alle Attribute näher erläutert (siehe Abschn. A.4).

```
1 ENTITY IfcElement
2  SUPERTYPE/SUBTYPE [...]
3  INVERSE
4   ConnectedTo : SET [0:?] OF IfcRelConnectsElements FOR RelatingElement;
5   ConnectedFrom : SET [0:?] OF IfcRelConnectsElements FOR RelatedElement;
6   [...]
7 END_ENTITY;
```

Auflistung 3.6: Definition inverser Attribute zwischen IfcElement und IfcRelConnectsElements, Auszug aus dem IFC4 Schema (buildingSMART International Ltd, 2013c), EXPRESS-G-Diagramme

In Instanzdateien nach ISO-10303-21 werden ausschließlich direkte Attribute explizit aufgeführt. Dies gilt ebenso für Gebäudemodelle im ifcXML-Format. Falls inverse Attribute benötigt werden, müssen die entsprechenden Referenzen in der verarbeitenden Applikation aufgebaut werden.

Um objektivierte Beziehungen auszuwerten sind inverse Attribute hilfreich, da die Navigation von einer konkreten Repräsentation für den Endanwender natürlicher erscheint als von einem abstrakten Beziehungsobjekt aus. Durch Kombination einer domänenspezifischen Verarbeitung von objektivierten Beziehungen und inversen Attributen wird im entwickelten BIM-Analysesystem die Navigation von der konkreten Repräsentation unterstützt (siehe Deassociater-Operator in Kap. 6.6.1).

3.3.6 Merkmalsliste

Ein *Property Set* (dt. *frei definierbare Merkmalsliste*) wird genutzt, um semantisch zusammenhängende Eigenschaften in einem Container zu hinterlegen. Die Nutzung von IfcPropertySets gegenüber der Eigenschaftsdefinition auf Entitätsebene hat folgende Auswirkungen:

- Die Entität erhält weniger unbelegte Eigenschaften, da nicht relevante Aspekte in Form ganzer Merkmalslisten entfallen und der Entität nicht zugeordnet werden.
- Die Navigation zu einer Eigenschaft einer Entität ist aufwendiger. Dies kann durch einmalige Vorprozessierung gemindert werden.

- Über benutzerdefinierte Merkmalslisten kann das Datenmodell ohne eigentliche Schemaanpassung individualisiert werden. Dadurch werden national angepasste und projektbezogene Austauschszenarien ermöglicht. Eine Erweiterung und regelmäßige Anpassung des IFC-Schemas mit einer allumfassenden Eigenschaftsdefinition für Entitäten ist innerhalb des langwierigen Spezifizierungsprozesses hingegen zu unflexibel.
- Da sich benutzerdefinierte Merkmalslisten dem Schema entziehen, ist keine Validierung gegenüber dem Schema möglich.

Die IFC Version 4 beinhaltet 408 vordefinierte Merkmalslisten. Diese werden in der XML-basierten Sprache *Property Set Definition (PSD)* modelliert (Adachi, 2015). Für eine Eigenschaft kann ein Datentyp sowie Bezeichnungen und Beschreibungen in mehreren Sprachen vorgehalten werden. Auflistung 3.7 enthält die Definition der *IsExternal*-Eigenschaft der vordefinierten `WindowCommon`-Merkmalsliste für Fenster.

```
<PropertyDef ifdguid="cd1b6c00d21811e1800000215ad4efdf">
  <Name>IsExternal</Name>
  <Definition>Indication whether the element is designed for use in the
  exterior (TRUE) or not (FALSE). [...]</Definition>
  <PropertyType>
    <TypePropertySingleValue>
      <DataType type="IfcBoolean" />
    </TypePropertySingleValue>
  </PropertyType>
  <NameAliases>
    <NameAlias lang="de-DE">Außenbauteil</NameAlias>[...]
  </NameAliases>[...]
</PropertyDef>
```

Auflistung 3.7: Eigenschaftsdefinition IsExternal der WindowCommon-Merkmalsliste des IFC4-Schemas (BuildingSMART, 2017a)

Auflistung 3.8 zeigt die Verwendung der `WindowCommon`-Merkmalsliste in einer beispielhaften P21-Datei. Die IfcWindow-Instanz wird über das IfcRelDefinesByProperties-Beziehungsobjekt mit dem IfcPropertySet verknüpft. Die Merkmalsliste besteht hier aus den zwei Eigenschaften *IsExternal* und *ThermalTransmittance*.

```
#3903= IFCWINDOW('...',#29,'Fenster-004',$,$,#3160,#3899,'...',1.5,1.);
#3942= IFCPROPERTYSET('...',#29,'Pset_WindowCommon',$,(#3948,#3954));
#3946= IFCRELDEFINESBYPROPERTIES('...',#29,$,$,(#3903),#3942);
#3948= IFCPROPERTYSINGLEVALUE('IsExternal',$,IFCBOOLEAN(.T.),$);
#3954= IFCPROPERTYSINGLEVALUE('ThermalTransmittance',$,
IFCTHERMALTRANSMITTANCEMEASURE(1.4),$);
```

Auflistung 3.8: Demonstration des Konzepts für frei definierbare Merkmalslisten anhand einer beispielhaften P21-Datei

In der Diskussion der Vererbungshierarchie von IfcWallStandardCase wurde beschrieben, dass Typinformationen als IfcTypeObjects repräsentiert und mit Bauteilen verknüpft werden können (vergl. Abbildung 3.3). Die zusätzliche Typisierung über IfcTypeObjects ermöglicht eine weitere Spezialisierung von IFC-Klassen. So können beispielsweise zwei Gruppen von IfcWindow-Instanzen mit zwei unterschiedlichen IfcTypeObjects verknüpft werden. Hierdurch erhält jede Gruppen gegebenenfalls auch eine eigene Geometrierepräsentation. Im Kontext von Merkmalslisten ist entscheidend, dass IfcPropertySets von IfcTypeObjects referenziert werden können. Sind gleichnamige Eigenschaften sowohl mit dem entsprechenden IfcTypeObject als auch mit der konkreten Bauteil-Instanz verknüpft, ist der bauteilspezifische Wert dominant. Dies veranschaulicht Tabelle 3.2.

Eigenschaft	an IfcWindowTyp	an IfcWindow	finaler Wert
IsExternal	TRUE		TRUE
FireRating		30	30
SmokeStop	FALSE	TRUE	TRUE

Tabelle 3.2: WindowCommon-Merkmalsliste an Typ- und Instanz-Objekt

Merkmalslisten erlauben eine erweiterte Anpassungsfähigkeit des IFC-Modells an die im jeweiligen Projekt erforderlichen Informationen. Die IFC-interne Repräsentation der entsprechenden Daten ist für den Endanwender wiederum nicht ausschlaggebend. Daher sollten die durch Eigenschaftsmengen bereitgestellten Attribute ähnlich standardmäßiger Entitätseigenschaften auszuwerten sein. Gleichzeitig müssen Sonderfälle, wie das Auftreten identisch benannter Eigenschaften, in unterschiedlichen Eigenschaftsmengen berücksichtigt werden. Dies erfordert die Möglichkeit, Auswertungen auf ein bestimmtes Property Set einzugrenzen (siehe PropertyFilter in Kap. 6.6.1).

3.3.7 Repräsentation der Entitätsgeometrie

Im *Computer Aided Design (CAD)* wurden Austauschformate, wie die *Initial Graphics Exchange Specification (IGES)* und das *Drawing Exchange Format (DXF)* eingesetzt. Konzeptionell steht hierbei die explizite Geometriebeschreibung von Komponenten im Vordergrund (Reed, 1991; Autodesk, 1998). Die heute eingesetzte Produktmodellierung realisiert indes eine zusätzliche semantische Ebene. Im IFC-Schema erfolgt die geometrische und die semantische Modellierung voneinander getrennt und mit möglichst geringen gegenseitigen Abhängigkeiten. Die semantische Ebene stellt in der IFC-basierten Modellierung zudem den führenden Informationsbereich und die alleinige Schnittstelle zur Datenbasis dar.

Im Folgenden wird erläutert, wie die semantische Modellierung, ausgehend vom bereits besprochenen IfcProduct, mit der geometrischen Modellierung verknüpft wird. Abbildung 3.7

zeigt den Zusammenhang zwischen IfcProduct und IfcGeometricRepresentationItem, das als Generalisierung aller geometrischer Repräsentationen innerhalb des Produktmodells fungiert.

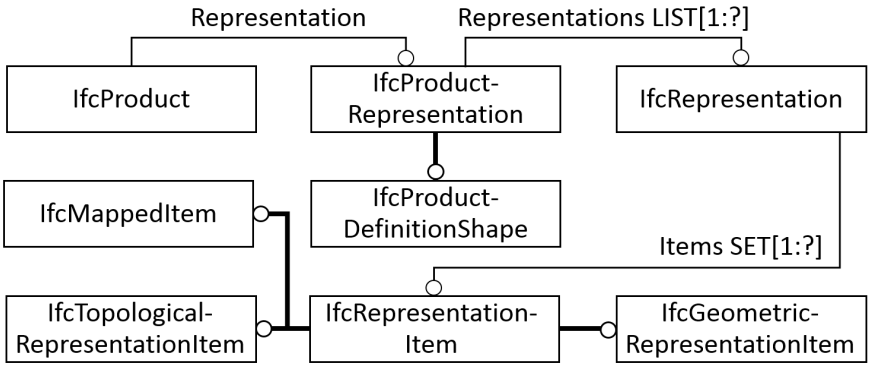


Abbildung 3.7: Das Zusammenspiel zwischen semantischer Entität (IfcProduct) und ihren Geometrierepräsentationen, EXPRESS-G-Diagramme

IfcProduct wird eine IfcProductRepresentation zugeordnet, wobei IfcProduct die existenzbestimmende Entität darstellt. IfcProductDefinitionShape als Subtyp von IfcProductRepresentation aggregiert geometrische, topologische und visuelle Eigenschaften. Dazu gehören neben der formgebenden Repräsentationen auch die optionale Aufteilung der Geometrie in Unterbereiche. Dadurch können beispielsweise unterschiedliche Materialien bestimmten Bereichen eines Produkt zugeordnet werden.

Von IfcRepresentation leiten IfcShapeRepresentation und IfcTopologyRepresentation ab. Diese referenzieren durch Subtypen von IfcRepresentationItem die eigentlichen geometrischen und topologischen Repräsentationen. Neben der textuellen Beschreibung der Repräsentationsart kann IfcRepresentation Werte für Renderoptionen und zur Ebenenzugehörigkeit aufnehmen.

IfcShapeRepresentation kann mit IfcRepresentationMap und IfcMappedItem zusammenwirken. Dadurch ist es möglich multiple, transformierte Instanzen einer geometrischen Grundform innerhalb einer Instanzdatei vorzuhalten. Auflistung 3.9 zeigt das beispielhafte Zusammenspiel von IfcColumn, Subklasse von IfcProduct, und einer mehrfach nutzbaren Geometrierepräsentation (IfcShapeRepresentation, Instanz #8).

```
#1= IFCCOLUMN('29j...',#,'Column...',$,'550x550',#,#2,'159790',.COLUMN.);
#2= IFCPRODUCTDEFINITIONSHAPE($,$,(#3));
#3= IFCSHAPEREPRESENTATION(#,'Body','MappedRepresentation',(#4));
#4= IFCMAPPEDITEM(#5,#6);
#5= IFCREPRESENTATIONMAP(#7,#8);
#6= IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#,1.,$);
#7= IFCAXIS2PLACEMENT3D(#,$,$);
#8= IFCSHAPEREPRESENTATION(#,'Body','SweptSolid',(#));
```

Auflistung 3.9: Zusammenspiel von IfcColumn und einer Geometrierepräsentation, nicht relevante Referenzen wurden mit dem #-Zeichen ersetzt

Nach der Betrachtung des Zusammenspiels zwischen semantischer und geometrischer Modellierung wird im Folgenden auf die unterschiedlichen Geometriepräsentationen im Kontext eines BIM-Transferformats eingegangen. Um Bauteile auch nach deren Erstellung effizient verändern zu können, wird in BIM-Anwendungen für Entwurf und Planung primär eine *prozedurale Geometriebeschreibung* genutzt. Anstelle geometrischer Grundelemente, wie Punkte oder Flächen bearbeiten zu müssen, können dadurch räumliche Repräsentationen auf hohem Abstraktionsniveau editiert werden.

Das IFC-Format dient dazu, Planungsdaten herstellerneutral zwischen unterschiedlichen Softwareprodukten auszutauschen. Dabei wird angestrebt, dass sich eingelesene Objekte wie interne, in der jeweiligen Applikation erzeugte Repräsentationen, verhalten. Dementsprechend realisiert das Datenmodell der IFC die Übertragung prozeduraler Geometriepräsentationen. Für Applikationen, in denen die Geometrie nicht verändert werden muss, können geometrische Körper auch in einer flächenbasierten Repräsentation übertragen werden.

Die prozedurale Modellierung basiert auf der Vorhaltung einer Historie an Erstellungs- und Bearbeitungsoperationen für jeden räumlichen Körper. Die Operationen können aus unterschiedlichen geometrischen Verfahren stammen. In der *Constructive Solid Geometry (CSG)* werden einfache Grundkörper auf Basis boolescher Operationen zu komplexen Körpern aggregiert (Requicha, 1980). Beim *Sweeping* wird ein geschlossenes Polygon an einem Pfad entlang geführt (Abdel-Malek et al., 2006). In einem *Rotationsverfahren* wird ein Polygon komplett oder teilweise um eine Achse gedreht (Lieu & Sorby, 2009, Kap. 6). Werden mehrere Flächen mit einer Hüllgeometrie umschlossen, wird der entstehende Körper als *Loft* und die Erstellungsmethode als *Lofting* bezeichnet (Filip & Ball, 1989). Die genannten Erstellungsmethoden sind in Abbildung 3.8 demonstriert. Für eine ausführliche Erörterung prozeduraler geometrischer Modellierung sei auf (Mäntylä, 1988) und (Monedero, 2000) verwiesen.

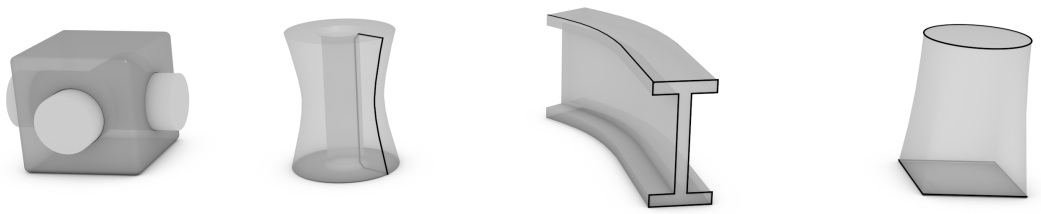


Abbildung 3.8: Prozedurale Modellierungsmethoden, v.l.n.r: CSG, Rotationsverfahren, Sweeping und Lofting

Neben dieser prozeduralen Beschreibungen, können geometrische Körper durch ihre Randflächen definiert werden. Dies wird im Englischen als *Boundary Representation (BRep)* bezeichnet. Ein Körper besteht hierbei aus Flächen und diese wiederum aus Kanten. Eine Kante wird durch zwei Knoten festgelegt. Die topologischen Informationen, welche Knoten eine bestimmte Kante ergeben und welche Kanten die Kontur einer Fläche definieren, wird in BRep-basierter

Modellierung meist durch eine Graphstruktur vorgehalten (vgl. Kapitel 2.2.3). Dabei existieren unterschiedliche Varianten, je nachdem welche topologischen Assoziationen vorgehalten werden. Beispiele sind die *Winged Edge*- und die *Half Edge*-Struktur (Baumgart, 1972, 1975).

Ein Spezialfall der BRep-basierten Geometriemodellierung ist die ausschließliche Verwendung von Dreiecken. Eine derartige triangulierte Oberflächenbeschreibung wird für numerische Berechnungen und Simulationen benötigt und kann direkt von Visualiserungshardware verarbeitet werden. Andererseits ist eine Bearbeitung von BRep-Körpern ausschließlich durch Veränderung der Grundelemente Flächen, Kanten und Knoten möglich.

Dagegen erhöht die Vorhaltung einer prozeduralen Beschreibung die Veränderbarkeit der Geometrie (Masuda, 1993, S. 2). Durch die Parametrisierung der Einzeloperationen ergeben sich vielfältige und effiziente Bearbeitungsmöglichkeiten. Um parametrische Geometrie visualisieren zu können, muss eine Konvertierung in eine BRep-basierten Repräsentation erfolgen. Dies ist mitunter aufwendig, jedoch Stand der Technik und in BIM-Anwendungn integriert. Eine Konvertierung von BRep in eine parametrisierte Form ist dagegen nicht ohne weiteres möglich und Gegenstand der Forschung (Shilane et al., 2004; Osada et al., 2002).

Das IFC-Produktmodell unterstützt die genannten Verfahren zur prozeduralen und BRep-basierten Repräsentation. Der zentrale Ausgangspunkt der Modellierung ist dabei IfcGeometricRepresentationItem. In Tabelle 3.3 werden die vielfältigen Spezialisierungen dieser Klasse erläutert.

Die Geometrierepräsentationen des IFC-Schemas sind für den Transfer bearbeitbarer Gebäudemodelle ausgelegt. Auf Basis der in Abschnitt 2.4 vorgestellten räumlichen Analysefunktionen von Geoinformationssystemen lässt sich folgern, dass die vorgehaltene Geometrie zusätzlich als Selektionskriterium und zur Analyse von Modellen eingesetzt werden kann. Die Konzepte der GIS-Analysen müssen dabei an die dreidimensionale Modellierung und den mitunter hohen Datenumfang der Gebäudemodelle angepasst werden (vgl. nachfolgendes TranslaTUM-Modell). Da sich aus allen Geometrierepräsentationen des IFC-Schemas explizite dreiecksbasierte BRep-Beschreibungen ableiten lassen, dient diese Form als Eingangsdatenquelle für die entwickelten räumlichen Algorithmen. Zudem lassen dreiecksbasierten Algorithmen in Kombination mit den in Abschnitt 2.5 besprochenen räumlichen Indizierungstechniken eine effiziente räumliche Analyse erwarten.

IfcGeometric-RepresentationItem Subklasse	Beschreibung
IfcBooleanResult	Das Ergebnis einer booleschen Operation zweier Körper, wobei Vereinigung, Schnitt und Differenz unterstützt werden.
IfcHalfSpaceSolid	Durch eine Ebene im Raum wird dieser in zwei Teilbereiche getrennt. Einer der beiden Bereiche bildet einen Körper, der als Operand in booleschen Operationen eingesetzt werden kann.
IfcSolidModel	Eine Repräsentation, die eine Unterscheidung zwischen dem Inneren, dem Rand und dem Äußeren eines Körpers erlaubt. Das Innere besteht dabei aus einer zusammenhängenden Menge an Punkten. Subklassen beinhalten BRep-, Sweep- und CSG-basierte Repräsentationen (IfcManifoldSolidBrep, IfcSweptAreaSolid, IfcCsgSolid)
IfcShellBased-SurfaceModel	Ein Flächenmodell auf Basis von offenen und geschlossenen Hüllflächen (IfcOpenShell, IfcClosedShell). Die einzelnen Teilflächen dürfen sich nicht überlagern.
IfcFaceBased-SurfaceModel	Ein vereinfachtes Flächenmodell. Im Gegensatz zum IfcShellBased-SurfaceModel können keine Hohlräume in Körpern beschrieben werden.
IfcCsgPrimitive3D	Abstrakte Superklasse für CSG-Grundkörper. Subklassen repräsentieren Blöcke (Quader), Pyramiden, Zylinder, Kegel und Kugeln.
IfcGeometricSet	Aggregation der geometrischen Grundelemente Punkte, Kurven und Flächen. Dabei werden keine topologischen Informationen vorgehalten.
IfcTessellatedItem	Superklasse für polygonale Repräsentationen mit einfacher Topologie. Durch die Subklasse IfcTriangulatedFaceSet können BRep-Modelle aus Dreiecken vorgehalten werden. Die Dreiecke werden dabei über Indizes in einer Punktliste definiert.

Tabelle 3.3: Spezialisierungen des IfcGeometricRepresentationItems innerhalb des IFC4-Schemas

3.3.8 Beispielmodell

Die Besprechung der IFC-basierten Modellierung wird im Folgenden durch die Präsentation eines beispielhaften Gebäudemodells abgeschlossen. Dieses wurde innerhalb des Projekts *TranslaTUM* konzipiert und repräsentiert ein modernes Klinikgebäude. Im Zuge einer Bachelorarbeit wurde das Modell in der Planungssoftware *Revit* erstellt und sowohl als P21- und als ifcXML-Datei exportiert (Brill, 2014). Tabelle 3.4 beinhaltet die Hauptmerkmale des Gebäudemodells und Abbildung 3.9 zeigt eine entsprechende 3D-Visualisierung.

Eigenschaft	Wert
Ersteller	Brill (2014)
Modellbezeichnung	TranslaTUM
Dateigröße STEP	22,8 MB
Dateigröße ifcXML	108,6 MB
IFC-Version	4
IFC-Entitäten	421085
Identitätsbehaftete IfcRoot-Instanzen	19775
Instanzen IfcProduct	4247
Instanzen IfcBuildingElement	3475
Instanzen IfcSpatialStructureElement	11
Metrische Einheit	Milimeter
Anzahl an Dreiecken nach BRep-Konvertierung	276298

Tabelle 3.4: Eigenschaften der IFC-basierten TranslaTUM-Modellierung

Abbildung 3.9: 3D-Visualisierung des TranslaTUM-Gebäudemodells (Brill, 2014)

3.4 Prozessbasierter Datenaustausch

In den vorherigen Abschnitten dieses Kapitels wurden die Konzepte einer IFC-basierten Modellierung erläutert. Ein entsprechendes Gebäudemodell vereint dabei alle relevanten Daten über den gesamten Lebenszyklus eines Gebäudes und dient idealerweise als alleinige Entscheidungsgrundlage für die Projektbeteiligten. Die Generalität des IFC-Datenmodells und unterschiedliche Modellierungsstrategien können dies jedoch verhindern.

Daher sind im Projektverlauf derzeit noch weitere fachliche Bestimmungen und technische Spezifikationen nötig. Dies ist besonders für eine zielführende Kollaboration zwischen Projektbeteiligten der Fall. Je nach betrachtetem Prozess unterscheiden sich die beteiligten Akteure, der thematische Rahmen und der Detailierungsgrad der auszutauschenden Daten. Dennoch können wiederkehrende Planungsprozesse auch über Projektgrenzen hinweg identifiziert und der jeweils stattfindende Datenaustausch inhaltlich festgelegt werden. Dazu muss Einigkeit über Planungsprozesse herrschen und Prozesse müssen strukturiert vorliegen. Dann können Inhalte übergeordneter Datenschemata mit den Prozessen in Verbindung gebracht werden.

3.4.1 Formalisierte Repräsentation von Planungsprozessen

Zur formalen Repräsentation von Planungsprozessen und entsprechender Modellinhalte dient das Konzept *Information Delivery Manual (IDM)*. Eine derartige Vorschrift bestimmt die Lokalisierung eines Prozesses bezüglich des Projektverlaufs und definiert den Zweck des Prozesses. Außerdem werden die agierenden Akteure beschrieben und die Daten, die diese erzeugen und konsumieren, festgelegt. Ein IDM dient außerdem zur Beschreibung wie die genutzten Daten durch Softwareanwendungen im Prozess verarbeitet werden können.

Aus Sicht der Fachanwender liefert der Ansatz eine verständliche Notation, um Prozesse des Bauwesens als *Prozessdiagramme* (engl. *process map*s) vorzuhalten. Als Bestandteil dieser Diagramme fungieren wiederverwendbare *Anforderungskataloge für den Informationsaustausch* (engl. *exchange requirement*s). Diese beschreiben die Voraussetzungen und Ergebnisse von Prozessen, um somit die Verlässlichkeit des Datenaustauschs und die Datenqualität zu erhöhen. Als übergeordnetes Ziel kann damit die Entscheidungsfindung in BIM-Projekten erleichtert und die Wertschöpfung beschleunigt werden. Für Softwareentwickler sind außerdem *Funktionsbausteine* (engl. *functional part*s) als dritter Bestandteil von IDMs relevant. Ein Baustein beinhaltet die Spezifikation des Datenaustauschs auf der technischer Ebene eines Schemas und legt somit benötigte Entitätsklassen und -attribute fest (Borrmann et al., 2015, S. 132). Abbildung 3.10 zeigt die beschriebene IDM-Architektur bestehend aus Prozessdiagramm, Anforderungskatalogen und Funktionsbausteinen.

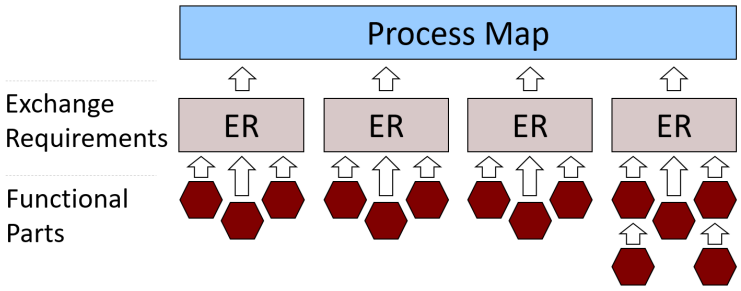


Abbildung 3.10: Dreistufige Architektur eines IDMs, nach (Wix & Karlshoej, 2010, S. 20)

Im Folgenden wird auf die unterschiedlichen Elemente des Modellierungsansatzes auf Basis von (Wix & Karlshoej, 2010) eingegangen.

Ein **Prozessdiagramm** beschreibt Tätigkeiten in den Grenzen eines thematischen Bereichs. Es dient dazu Klarheit über die Eigenschaften dieser Tätigkeiten und des zeitlichen Ablaufs zu gewinnen. Außerdem werden die beteiligten Akteure identifiziert und Aussagen über die benötigen und konsumierten Informationen getroffen.

Ein **Anforderungskatalog** definiert die Informationsmenge, die auf Grund der fachlichen Anforderungen zu einem bestimmten Zeitpunkt im Projekt verfügbar seien muss. Das IDM-Konzept wurde seitens buildingSmarts primär für die Nutzung mit dem IFC-Schema entwickelt. Anforderungskataloge können jedoch auch auf Basis weiterer standardisierter Datenmodelle, wie beispielsweise CityGML, erzeugt werden. Als Dokumentationsquelle für Fachanwender wie Architekten und Ingenieure beschreibt ein derartiger Katalog die zu liefernde bzw. zu erwartende Informationsmenge in nicht-technischer Notation.

Im Gegensatz dazu bezieht sich ein **Funktionsbaustein** auf ein konkretes, standardisiertes Informationsmodell wie die IFC. Der Baustein kapselt dabei eine einzelne Informationseinheit. Er stellt somit für sich selber ein eigenes Informationsmodell dar, ist aber gleichzeitig eine Untermenge des ursprünglichen Modells. In einem Funktionsbaustein kommen außerdem sogenannte *Konzepte* (engl. *concepts*) zum Einsatz. Ein IDM-Konzept ist ein Informationsfragment und legt grundlegende Funktionalität fest, wie beispielsweise die eindeutige Entitätsbezeichnung durch das GlobalId-Attribut. Ein Funktionsbaustein dient der Anwendungsentwicklung, um den prozessbasierten Datenaustausch durch geeignete rechnergestützte Werkzeuge zu unterstützten.

Die technische Umsetzung eines Anforderungskatalog ergibt ein Modell des Katalogs (engl. *exchange requirement model*). Das Modell ist versionsabhängig zum übergeordneten Datenschema. Daher kann ein Anforderungskatalog auf mehrere Modelle verweisen und somit unterschiedliche Versionen eines Schemas unterstützen. Zweck des Katalogmodells ist die Bereitstellung eines Teildatenschemas für einen wohldefinierten Informationsaustausch. Dieser ist definiert durch seinen Zweck, seinen Zeitpunkt im Projektablauf und kann entsprechend

seines Einsatzortes lokalisiert werden. Erstellt wird ein konkretes Katalogmodell durch die Aggregation der benötigen Funktionsbausteine

Zur Definition von Prozessdiagrammen dient eine erweiterte Version der *Business Process Modelling Notation (BPMN)*. Damit können die bereits beschriebenen Inhalte wie Akteure, Abfolgen von Informationsübergaben und Informationsmengen übersichtlich visualisiert werden. Zur Illustration ist im Anhang in Abschnitt A.5 ein Prozessdiagramm bezüglich einer BIM-basierten Energieanalyse enthalten.

Für eine weiterführende Diskussion über den prozessbasierter Datenaustausch im Bauwesen sei auf (Wix & Karlshoej, 2010; Richard et al., 2012) und (Borrmann et al., 2015, Kap. 4 u. 7) verwiesen. Die Business Process Modelling Notation ist in (Wix, 2007) beschrieben. Zur rechnergestützten Verarbeitung der in einem Katalogmodell genutzten Modellsichten wurde ein eigenständiges Datenformat spezifiziert. Darauf wird im nachfolgenden Abschnitt näher eingegangen.

3.4.2 Definition von Modellsichten

Eine *Model View Definition (MVD)* beschreibt eine Untermenge des IFC-Schemas und setzt Anforderungskataloge für den Informationsaustausch um. Jede Modellsicht besitzt einen klar definierten fachspezifischen Geltungsbereich und kann sich auf eine oder mehrere IDMs beziehen. Außerdem beschreibt eine Sicht Strukturen und Beziehungen von IDM-Konzepten und hält entsprechende Implementierungsregeln vor.

Damit liefert ein MVD klare Einschränkungen des weitläufigen IFC-Schemas und stellt die technische Umsetzung eines IDM-basierten Planungsprozesses dar. Auf Basis von Modellsichten wird zudem der Zertifizierungsprozess von IFC-Schnittstellen in BIM-Anwendungen durchgeführt (Richard et al., 2012, S. 5). Parallel zum IFC-Schema werden eine Reihe vordefinierter Modellsichten für unterschiedliche Prozesse bereitgestellt, die im Folgenden beschrieben werden.

IFC4 Reference View: Diese Modellsicht der IFC-Version 4 realisiert eine Datenübergabe für BIM-Anwendungen, die keine Geometrieänderungen unterstützen (BuildingSMART, 2014b).

IFC4 Design Transfer View Ebenfalls für die IFC-Version 4 konzipiert, ermöglicht diese Sicht die Datenweitergabe an Applikationen, die Änderungen an einem Modell vornehmen müssen. Das schließt das Editieren von Geometrierepräsentationen mit ein (BuildingSMART, 2014a).

Coordination View Version 2.0: Die Sicht ermöglicht den Transfer von editierbaren Modellen zwischen planerischen und ingenieurtechnischen Anwendungen. Die Version 2.0

wurde mit IFC-Version 2x3 eingeführt und stellt die aktuell am häufigsten unterstützte Modellsicht in BIM-Anwendungen dar (BuildingSMART, 2009a).

Space Boundary Addon View: Diese Sicht gestattet es, Modelle an Applikationen weiterzugeben, die detaillierte geometrische und thermale Informationen über Räume und Zonen benötigen. Diese Modellsicht ist eine Erweiterung des Coordination View 2.0 (BuildingSMART, 2010).

Basic Facility Management Handover View: Diese Sicht für IFC-Version 2x3 wurde etabliert, um die Wiederverwendung vorhandener Planungsdaten im *Computer Aided Facility Management (CAFM)* zu ermöglichen (BuildingSMART, 2009b).

Structural Analysis View: Diese Sicht erlaubt, basierend auf der IFC-Version 2x3, die Übergabe eines Modells an eine Berechnungssoftware für statische Analysen (BuildingSMART, 2007).

Zur formalen Spezifikation von Modellsichten wurde das *mvdXML*-Format entwickelt (Chipman et al., 2013). Auflistung 3.10 zeigt eine stark vereinfachte mvdXML-Datei mit den Konzepten `Aggregation` und `Classification` bezüglich Trägern und Stützen (Weise, 2015).

```
1  <mvd:mvdXML name="buildingSMART international MVD's">
2    <mvd:Templates>
3      <mvd:ConceptTemplate  name="Aggregation">
4        <mvd:SubTemplates>
5          <mvd:ConceptTemplate  name="Decomposition" />
6          <mvd:ConceptTemplate name="Composition" />
7        </mvd:SubTemplates>
8      </mvd:ConceptTemplate>
9      <mvd:ConceptTemplate name="Classification" />
10   </mvd:Templates>
11   <mvd:Views>
12     <mvd:ModelView name="CoordinationView_V2.0">
13       <mvd:ExchangeRequirements>
14         <mvd:ExchangeRequirement name="Architecture" />
15         <mvd:ExchangeRequirement name="BuildingService" />
16       </mvd:ExchangeRequirements>
17       <mvd:Roots>
18         <mvd:ConceptRoot name="Beam" applicableRootEntity="IfcBeam" />
19         <mvd:ConceptRoot name="Column" applicableRootEntity="IfcColumn" />
20       </mvd:Roots>
21     </mvd:ModelView>
22   </mvd:Views>
23 </mvd:mvdXML>
```

Auflistung 3.10: Prinzipieller Aufbau einer mvdXML-Datei, Beispiel zu den Konzepten Aggregation und Classification (Stützen und Trägern), nach (Weise, 2015)

Die Struktur einer mvdxml-Instanzdatei ergibt sich aus Konzeptvorlagen (ConceptTemplates) und Hauptkonzepten (RootConcepts), die in einer Modellsicht (ModelView) enthalten sind. Eine Konzeptvorlage repräsentiert ein wiederverwendbares Konzept und kann aus einer Hierarchie aus Unterkonzepten gebildet werden. Anforderungskataloge für den Informationsaustausch werden als ExchangeRequirement-Elemente abgebildet. Eine Modellsicht enthält die Hauptkonzepte und verweist auf die relevanten Anforderungskataloge und Konzeptvorlagen.

Da Modellsichten auf einer Reduktion der zu übertragenden IFC-Entitäten beruhen, kann eine Filterung mit Hilfe einer Anfragesprache zur Realisierung derartiger (benutzerdefinierter) Sichten beitragen. Dabei kommen Typ- und Attribut-Filter zum Einsatz, wobei assoziierte Eigenschaftsmengen (IfcPropertySets) in die Betrachtung mit eingeschlossen werden müssen. Auf ein Konzept zur MVD-basierten Filterung wird in Kapitel 5.3 eingegangen.

3.5 Terminplanung

In einer BIM-basierten Planung definiert derzeit ein semantisch-räumliches Modell das zu verwirklichende Bauwerk. Die Betrachtung der zeitlichen Dimension in einem Bauvorhaben mündet hingegen in eine Terminplanung, die maßgeblich zur wirtschaftlichen Fertigung des geplanten Gebäudes beiträgt. Im Folgenden wird einleitend auf die Terminplanerstellung eingegangen. Danach wird die angestrebte Kombination terminlicher und modellbasierter Planung besprochen. Abschließend wird die in der IFC4 realisierte 4D-Modellierung vorgestellt.

3.5.1 Terminplanerstellung

Wie die räumliche Planung erfolgt auch die Terminplanung in aufeinander aufbauenden Detaillierungsstufen (Hofstadler, 2007, S. 48 u. 67). In der Grobplanung bezieht sich die zeitliche Betrachtung auf ganze Bauwerke oder Teillose. Die Ergebnisse dienen unter anderem zur Erstellung eines Generalablaufplans, der Angebotsvorbereitung und um die Gesamtdauer des Projektes abschätzen zu können. Auf Basis der Grobplanung erfolgt nachgelagert die Feinplanung. In dieser werden sequenziell und parallel ablaufende Produktionsprozesse behandelt. Im Projektverlauf steigt die Detaillierung der Terminplanung kontinuierlich an. Dazu werden Prozesse in jedem nachfolgenden Detaillierungsgrad durch genauer abgebildete Unterprozesse ersetzt.

Um einen Terminplan zu erstellen, benötigt es Wissen über die Dauer einzelner Prozesse innerhalb des Bauvorhabens. Der Prozessbegriff hat je nach Fachbereich unterschiedliche Be-

deutungen. Enge (2010, S. 13) stellt folgende Definition im Kontext der Bauablaufplanung auf:

> „Ein Prozess stellt eine Zerlegung eines komplexen Arbeitsvorhabens in eine Menge an Vorgängen und Zuständen dar, für die eine Abfolge definiert ist. Vorgänge stellen zeitfordernde Geschehen dar. Ihnen werden Ressourcen, Maschinen und Material zugeordnet. Zustände stellen Ereignisse dar, die einen Vorgang auslösen und die das Ergebnis eines Vorgangs beschreiben."

Zur Ermittlung von Zeitintervallen für Prozesse werden in (Tulke, 2010, S. 15) vier Methoden angegeben, die im Folgenden nach ihrem zunehmenden Detaillierungsgrad vorgestellt werden:

- Eine grobe Zeitschätzung kann nach Erfahrungswerten unter Zuhilfenahme der projektspezifischen Randbedingungen geschehen. Diese Methode wird häufig für Zeitintervalle angewendet, die nicht direkt mit Produktionsprozessen verknüpft sind, sondern sich auf detaillierte Planungs- und Vorarbeiten beziehen.
- Standardmäßig wird mit Zuhilfenahme des Terminplans die Kostenermittlung durchgeführt. Da in frühen Entwurfsphasen keine detaillierten Prozessdaten existieren, wird in dieser Methode diese Abhängigkeit vertauscht. Durch die auf Gewerke aufgeteilten Kostenschätzungen können Zeitintervalle auf einer groben Detailierungsstufe abgeleitet werden.
- Aus der Entwurfsplanung können geometrische Richtgrößen wie Geschossflächen für die Terminplanung ermittelt werden. Unter Berücksichtigung der Projektgegebenheiten kann dadurch auf die benötigten Zeiten der entsprechenden Ausführungsprozesse geschlossen werden.
- Ist die Planung bis zur Ebene der herzustellenden Bauteile fortgeschritten, können Bauteilmengen berechnet werden. Auf Basis von Zeitbedarfsfaktoren für die jeweiligen Arbeitsschritte kann aus den Mengendaten ein detaillierter Terminplan erstellt werden.

Auf Basis von Zeitdauern, Vor- und Nachfolge-Beziehungen sowie Zwangsbedingungen wie die Ressourcen-Verfügbarkeit werden manuell oder teilautomatisiert Terminpläne in unterschiedlicher Granularität erstellt. Die Qualität eines Terminplans umfasst dabei eine Reihe von Eigenschaften. Dazu gehört die Gesamtdauer des Plans, die Auslastung von Ressourcen und die Konfliktfreiheit bei deren Verwendung. Zur Analyse eine Terminplans hinsichtlich dessen Robustheit und der erzielten Ressourcenauslastung können Systeme eingesetzt werden, die auf der *Critical Path Method* und der *Discrete Event Simulation* basieren (Dori et al., 2012).

Diese Systeme erlauben jedoch nicht die Generierung eines optimalen Terminplans, da auf Grund der Komplexität des Problems in der Regel heuristische Optimierungstechniken ein-

gesetzt werden. Anwendung finden beispielsweise Verfahren auf Basis *genetischer Algorithmen* und des *Simulated Annealings* (Swisher et al., 2000). Eine Weiterentwicklung stellt die *Constraint-based Discrete Event Simulation* dar, bei dem Fachwissen über die einzelnen Prozesse als Zwangsbedingungen abgebildet werden. Dadurch können auch Fälle, in der eine doppelte Ressourcenbelegung normalerweise eine Zufallsentscheidung auslöst, dennoch optimiert werden (König et al., 2007).

3.5.2 Kombination terminlicher und modellbasierter Planung

In der heutigen Praxis wird die Terminplanung standardmäßig getrennt von einem Gebäudemodell vollzogen. Dabei kommen Softwarewerkzeuge aus dem Bereich des Projektmanagments wie *Microsoft Project* und *Synchro Pro* zum Einsatz. Somit sind beide Planungsumgebungen bislang voneinander getrennt, obwohl das Gebäudemodell alle relevanten Ausgangsdaten für die Terminplanung beinhaltet. Daten werden deswegen häufig manuell aus dem Gebäudemodell entnommen und in Werkzeuge der zeitlichen Planung eingepflegt. Die Vorgehensweise ist nicht nur fehleranfällig sondern erfordert eine zeitaufwendige Nachbearbeitung bei Änderungen.

Im Idealfall wird das Gebäudemodell und der Terminplan des Bauablaufs in einem 4D-Modell zusammengeführt und eine Animation des Bauvorhabens erstellt. Diese kann genutzt werden, um raum-zeitliche Konflikte wie beispielsweise überlappende Arbeitsbereiche bereits vor Ausführungsbeginn zu erkennen. In (Akinci et al., 2002) wird eine Taxonomie für raum-zeitliche Konflikte definiert. Außerdem werden Methoden zur automatischen Erkennung derartiger Konflikte präsentiert. Weitere automatische Ansätze für die Konflikterkennungen werden von Zülch & Stock (2010) und Moon et al. (2014) beschrieben. Einschränkungen bestehen bei den vorgestellten Ansätzen in den verwendbaren Geometrierepräsentationen, bei denen es sich um AABBs bzw. Arbeitsbereiche von Kränen handelt.

Die Integration des Bauablaufs in die modellbasierte Planung ist seit längerem Gegenstand der Forschung (McKinney & Fischer, 1998). Ziel ist es, detaillierte Terminpläne auf Ebene der Bauteile in das Gebäudemodell zu integrieren und dabei Prozesse und Ressourcen mit abzubilden. Dabei erfolgt die zeitliche Definition des Bauprozesses derzeit noch getrennt von der architektonischen Planung und der Kostenermittlung (Tulke et al., 2008b). In der Terminplanung werden Prozesse, in der räumlichen Modellierung hingegen Bauteile mit ihren geometrischen Repräsentationen betrachtet. Letztendlich wird eine Verknüpfung des Produktmodells mit dem Prozessmodells des Terminplans angestrebt (siehe Abbildung 3.11).

Als teilweise problematisch erweist sich die Zuordnung von Prozessen zu den räumlichen Repräsentationen von Bauteilen. In (Tulke et al., 2008a, S. 2) werden fünf Konstellationen identifiziert, wobei der zweite und fünfte Fall nur abgebildet werden können, wenn Bauteile entsprechend der Prozessinformation zusätzlich unterteilt werden (siehe Abbildung 3.12).

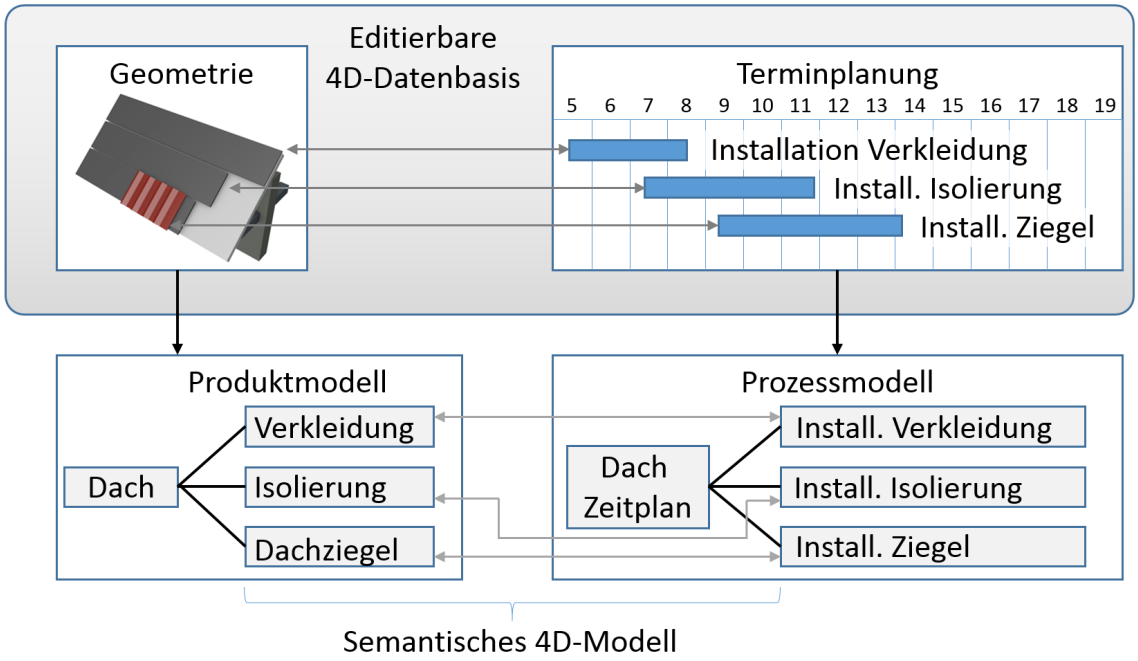


Abbildung 3.11: Angestrebte Verknüpfung des Produktmodells mit dem Prozessmodell des Terminplans, nach (McKinney & Fischer, 1998)

Diese geometrische Zerlegung auf Basis von Prozessdaten wird in der derzeitig eingesetzten BIM-Software nicht unterstützt.

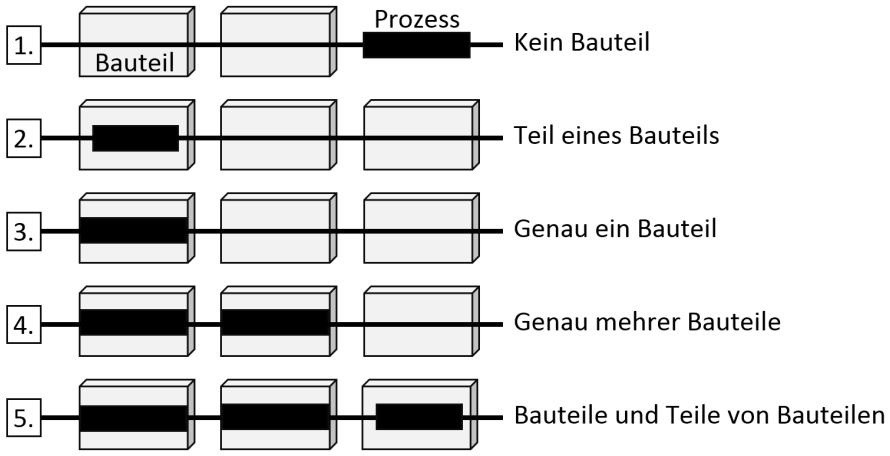


Abbildung 3.12: Zuordnung von Prozessen und Bauteilen, nach (Tulke et al., 2008a)

Tulke et al. (2008a, Kap. 2) stellen daher eine Methode zur automatischen Zerlegung von Bauteilen vor. Der eingesetzte Algorithmus wurde ursprünglich für boolesche Operationen einer CSG-Modellierung entwickelt (Laidlaw et al., 1986). Innerhalb einer detaillierten Terminplanung wird ein Gebäude häufig in Zonen gegliedert. Daraus wird gefolgert, dass Bauteile auf Basis der dreidimensionalen Zonengeometrie zerlegt werden müssen. Abbildung 3.13 zeigt

die Zerlegung einer Wand in drei einzelne Bauteile auf Grund ihrer räumlichen Interaktion mit der Zone. Des Weiteren schlagen Tulke & Hanff (2007, Kap. 3.1) vor, eine Anfragesprache, hier die *Structured Query Language*, zur regelbasierten Verknüpfung von Bauteilen und zeitlich modellierten Aufgaben zu nutzen.

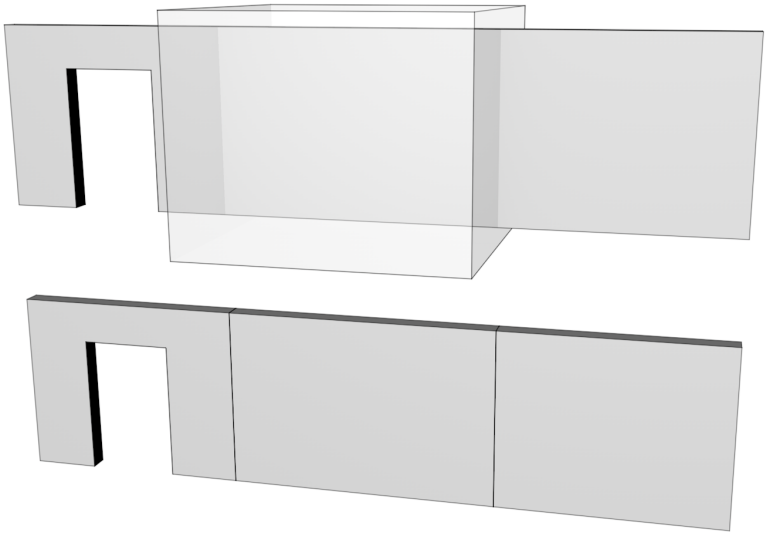


Abbildung 3.13: Zerlegung eines Bauteils in Unterelemente auf Basis von Zonen, nach (Tulke et al., 2008a)

3.5.3 Raum-zeitliche Modellierung

Das IFC-Objektmodell erlaubt eine assoziative Verknüpfung von Bauteilen mit der bereits vorgestellten IfcTask-Entität (siehe Abbildung 3.1). Diese hält durch eine referenzierte IfcTaskTime mehrere Zeitperioden vor. In Abbildung 3.14 sind die IFC-Typen zur Verknüpfung von Bauteilen und Aufgaben dargestellt.

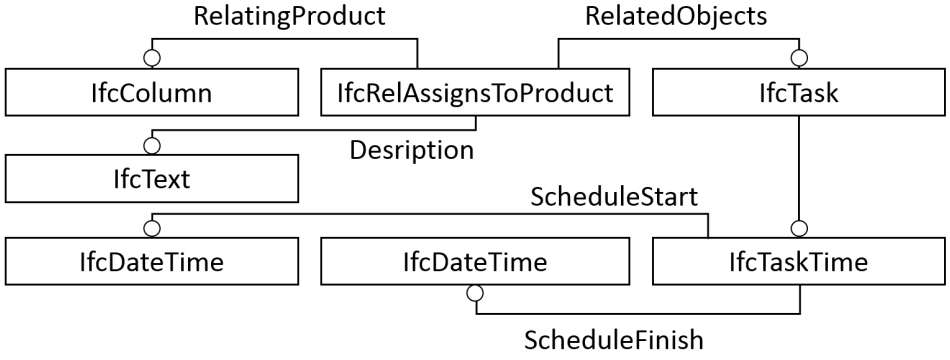


Abbildung 3.14: Verknüpfung von Bauteilen mit semantisch unterschiedlichen Zeitperioden im IFC-Schema, EXPRESS-G-Diagramm

Ein Bauteil (IfcBuildingElement) wird über ein objektiviertes Beziehungsobjekt (IfcRelAssignsToProduct) mit einem IfcTask verknüpft. Eine Instanz der IfcTask-Klasse steht hierbei für eine projektspezifische Aufgabe, beispielsweise für die Erstellung der Bewehrung eines Ortbeton-Bauteils. In der referenzierten IfcTaskTime können unterschiedliche Zeitperioden über die im Folgenden genannten Attribute vorgehalten werden. Dies beinhaltet eine Zeitperiode, in der die Aufgabenausführung ohne negative Effekte für den globalen Bauablauf möglich ist (*EarlyStart*, *LateStart*, *EarlyFinish*, *LateFinish*). Außerdem kann eine Periode über die konkret geplanten Zeiten vorgehalten werden (*ScheduleStart*, *ScheduleFinish*). Während des Bauablaufes muss der Baufortschritt erfasst und kontrolliert werden. Um derartige Daten wieder in das IFC-Modell einfließen zu lassen, existiert eine weitere Zeitperiode (*ActualStart*, *ActualFinish*). Die drei unterschiedlichen Perioden sind in Abbildung 3.15 dargestellt.

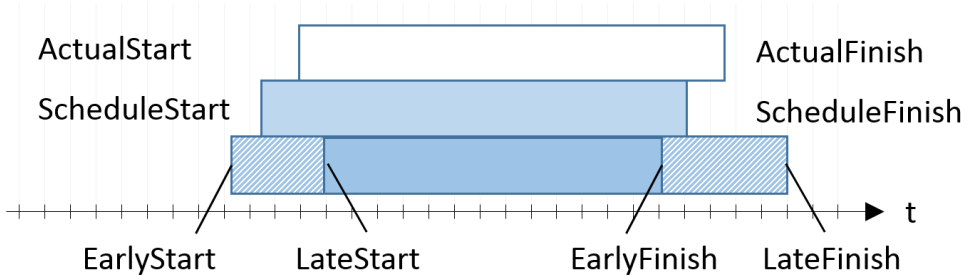


Abbildung 3.15: In einer IFC-basierten Modellierung abbildbare Zeitperioden des Bauablaufs

Aufgaben können als Elemente einer Teil/Ganzes-Hierarchie vorgehalten werden. Ein IfcTask wird dazu mit IfcRelNests über dessen *RelatingObject*-Attribut als übergeordnete Aufgabe definiert. Über *RelatedObjects* referenziert die objektivierte Relation die jeweiligen Unteraufgaben. Eine lineare Abfolge zwischen einzelnen IfcTasks wird über die objektivierte Relation IfcRelSequence modelliert. Dabei verweist das *RelatingProcess*-Attribut auf die vorgelagerte Aufgabe. Die nachfolgende Aufgabe wird über das *RelatedProcess*-Attribut referenziert. Abbildung 3.16 zeigt die beiden Modellierungsansätze als EXPRESS-G-Diagramme.

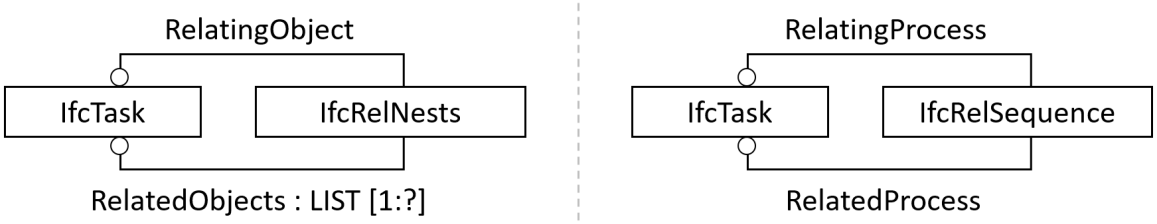


Abbildung 3.16: Komposition und Sequenzierung aufgabenbezogener IfcTask-Entitäten, EXPRESS-G-Diagramme

Die IFC-basierte Beschreibung des Bauablaufs ist als eine domänenspezifische objektorientierte Variante zeitlicher Modellierung anzusehen. Bedingt durch den Planungsbezug steht dabei die explizite Verknüpfung von Objekten mit zeitlichen Daten im Vordergrund. Der IfcTaskTime-Typ wird genutzt um die Zeitspannen, in denen Bauteile erstellt bzw. installiert werden vorzuhalten. Erstellung und Installation verändern die Eigenschaften eines Bauteils über eine Zeitspanne hinweg. Dem gegenüber steht die Betrachtung der zeitlichen Veränderung des Datenbestands. Eine Abbildung dieser Veränderung durch Transaktionen kann in Einzelfällen aufschlussreich sein, ist jedoch nicht Bestandteil der eigentlichen Modellierung. Daher wird die Transaction Time in einem BIM-basierten Ansatz nicht vorgehalten. Die Komplexität der zeitlichen IFC-Modellierung kann im Vergleich zu den Anforderungen an ein zeitliches GIS als moderat eingestuft werden.

3.6 Fazit

Das herstellerneutrale und offene IFC-Schema stellt eine umfangreiche Strategie zur computerbasierten Repräsentation von Bauwerke dar. Der Ansatz unterscheidet sich stark zu den zuvor eingesetzten Methoden. In diesen wurde Geometrie als die primär auszutauschende Information angesehen.

Das zentrale Element der IFC-Modellierung ist hingegen die Entität. Damit wird ein komplexes Objekt im Sinne der objektorientierten Modellierung bezeichnet. Es besitzt einen festen Typ, beschreibende Attribute und eine eindeutige Identität. Das Entitätskonstrukt wurde in der IFC mit EXPRESS- und domänenspezifischen Ansätzen erweitert. Dazu gehören unter anderem objektivierte Relationen, inverse Attribute und Merkmalslisten. Geometrische Repräsentationen sind nach wie vor ein unverzichtbarer Teil des Datenaustausches. Dies ist in der IFC durch die zahlreichen Repräsentationsformen für räumliche Daten ersichtlich. Geometrische Entitäten sind hier jedoch Teil einer objektorientierten, in Schichten gegliederten Modellierung und nur über höherliegende semantische Entitäten zugänglich.

Das IFC-Schema unterstützt zudem die Vorhaltung zeitlicher Daten der Terminplanung. Durch objektivierte Relationen lassen Realisierungszeiträume als verknüpfte Elemente mit Vor- und Nachfolger darstellen. Außerdem können aufgabenbezogene Zeitintervalle an semantische Entitäten mit Raumbezug gebunden werden. Da sich diese Intervalle auf die Erstellung und Installation dieser räumlichen Entitäten beziehen, erscheint die Kombination dieser Daten sinnvoll.

Durch die aufgezeigten Konzepte können Gebäudemodelle mit hohem Informationsgehalt erstellt und ausgetauscht werden. Anderseits werden die enthaltenen Informationen derzeit nicht für einen formalen, automatisierbaren Zugang zum Modell eingesetzt. Im Hinblick auf die feingranulare Modellierung und dem Datenumfang von Gebäudemodellen wird dies als problematisch eingestuft. Potential für eine höhere Modellqualität und effizientere Arbeitsweise bleibt ungenutzt.

Gleichzeit ist zu erwarten, dass verfügbare Ansätze zur Datenextraktion und -analyse für die Verarbeitung von IFC-Daten nicht geeignet sind. Die feingranulare Modellierung des Schemas und der inhaltliche Umfang bringen für entsprechende Systeme hohe Anforderungen mit sich.

Kapitel 4

Anwendung bestehender Analysekonzepte auf Gebäudemodelle

Datenbanken werden für die Analyse und Weiterverarbeitung ausgedehnter Datenbestände und als Bestandteil datenintensiver Anwendungen eingesetzt. Wegen des Umfangs, den Gebäudemodelle erreichen können, nutzen auch BIM-Anwendungen teilweise Datenbanken als interne Komponenten. Hierbei stehen eine strukturierte Datenhaltung und ein effizientes Speichermanagement im Vordergrund. Datenbankbasierte Analysefunktionen, wie in Geoinformationssystemen üblich, werden dem Nutzer von BIM-Anwendungen nicht bereitgestellt. Dies kann damit erklärt werden, dass die verfügbaren Konzepte für die Verarbeitung von Gebäudemodellen nicht geeignet sind.

Dennoch liefern Datenbanken mit ihren dedizierten Modellierungsstrategien und den darauf agierenden formalen Anfragesprachen Grundlagen für Analysen im BIM-Bereich. Im Folgenden werden daher Prinzipien der entsprechenden Datenmodellierungen und Sprachen vorgestellt. Anhand von Musteranfragen wird die Anwendung der unterschiedlichen Konzepte zur Analyse von BIM-Daten erörtert. Im letzten Teil des Kapitels wird außerdem auf domänenspezifische Anfragesprachen eingegangen.

4.1 Formale Anfragen

Building Information Modeling basiert auf der Nutzung einer umfassenden, computerinterpretierbaren Bauwerksrepräsentation. Diese wird für alle planerischen und betrieblichen Interaktionen mit dem Bauwerk genutzt. Wie im letzten Kapitel gezeigt, wird hierzu eine feingranulare Modellierung eingesetzt, die sich in vier Teilbereiche gliedern lässt. So sind für alle Entitäten eines Gebäudemodells Typ- und Attributwerte verfügbar (Teilbereich **Typisierung**). Der zweite Bereich ergibt sich durch explizit modellierte bzw. durch Prozessierung ableitbare Rela-

tionen zwischen Entitäten (Teilbereich **Relationen**). Durch die dreidimensionale Bauteilgeometrie wird der dritte Bereich des Modells definiert (Teilbereich **Geometrie**). Des Weiteren bildet der zeitliche Bauablauf mit den geplanten Installations- und Erstellungszeiträumen von vorgehaltenen Bauteilen einen eigenen Bereich (Teilbereich **Zeit**). Abbildung 4.1 illustriert die genannten vier Teilbereiche eines Gebäudemodells.

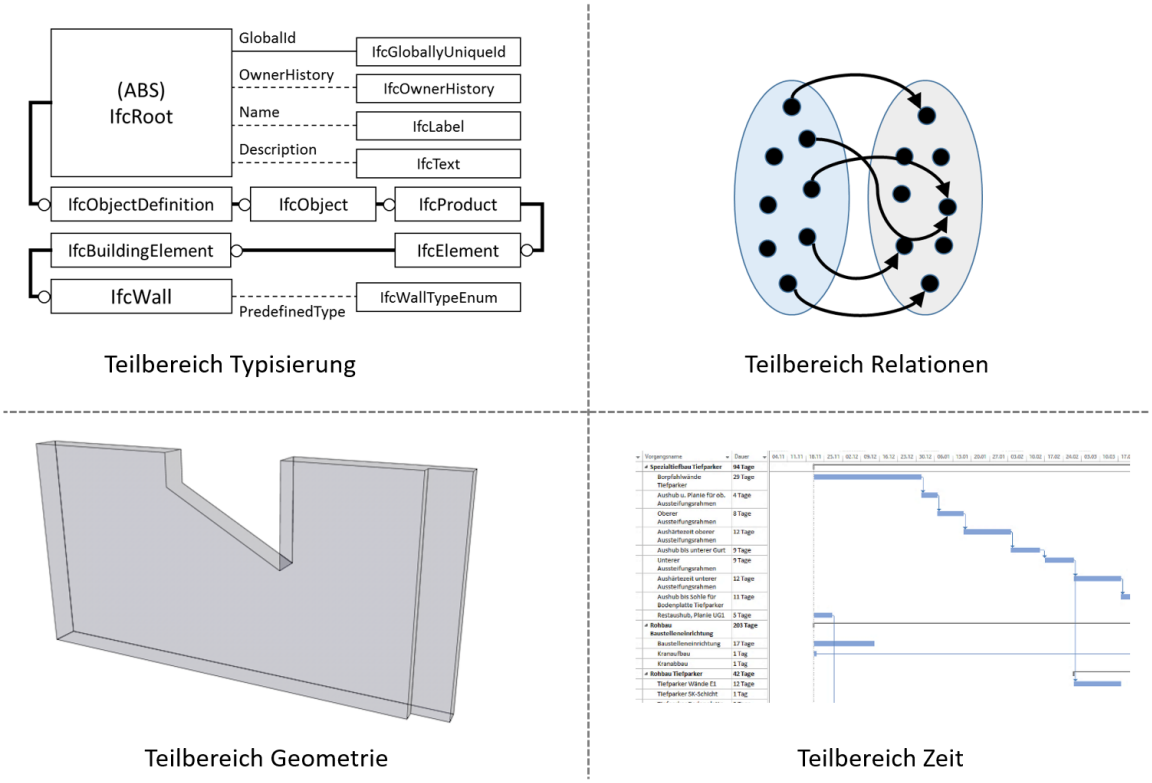


Abbildung 4.1: Konzeptionelle Aufteilung eines Gebäudemodells in vier Teilbereiche

Auf Grund dieser umfassenden Datenmodellierung ergeben sich vielfältige Anwendungsfälle für eine BIM-Anfragesprache. Hierbei führt die Nutzung einer formalen Sprache zu mehreren Vorteilen. So basieren die Operatoren einer Anfragesprache auf festen Definitionen. Dadurch liefert die Ausführung derartiger Operatoren eindeutige Ergebnisse, die nicht interpretationsbedingt zwischen unterschiedlichen Bearbeitern abweichen. Durch formale Anfragen können zudem Bearbeitungsfehler reduziert werden. Derartige Fehler sind gerade bei datenintensiven Gebäudemodellen wahrscheinlich, beispielsweise wenn eine zu modifizierende Bauteilkonstellation auf Grund des Informationsumfangs übersehen wird.

Ein aus wirtschaftlicher Sicht wichtiger Vorteil einer automatisierbaren Modellschnittstelle ist die damit verbundene einfache Wiederholung komplexer Selektionsschritte. Gerade im Kontext einer ausgedehnten Datenbasis und wiederkehrender Aufgaben kann hierbei von einer hohen Effizienzsteigerung ausgegangen werden.

Zu den grundlegenden Einsatzgebieten einer BIM-Anfragesprache gehören die Ausführung von *Ad-hoc-Anfragen*, die *Modellvalidierung* und die *Erstellung von Teilmodellen*. Im Folgenden werden diese Einsatzgebiete näher beschrieben.

Ad-hoc-Anfragen bezeichnen sprachbasierte Filterungen und Analysen, die im Zuge fortlaufender Projektarbeit ausgeführt werden müssen. Dabei steht die Reduktion der Datenbasis auf den momentan benötigten Umfang im Vordergrund, ebenso die Inspektion bestimmter Modellbereiche und Entitätskonstellationen. In 4D-Gebäudemodellen spielen außerdem raumzeitliche Betrachtungen eine Rolle. Beispiele für einfache Ad-hoc-Anfragen sind:

- „Selektiere alle Bauteile des 3. Stockwerks."
- „In welchen Räumen befinden sich Bauteile der Klimaanlage?"
- „Selektiere alle Wände, die in Kalenderwoche 5 errichtet werden sollen."

Die **Validierungen eines Gebäudemodells** mit Hilfe einer Anfragesprache kann nochmals in drei Varianten eingeteilt werden. In einer grundlegenden Validierung während des Imports wird das Vorhandensein der benötigten Entitäten und deren Eigenschaften geprüft. Dies kann auf Basis einer Model View Definition erfolgen. Eine erweiterte Analyse ist erforderlich, um die räumlich-semantische Konsistenz eines Gebäudemodells zu validieren. Zu entsprechenden Inkonsistenzen kann es kommen, da IFC-Modelle bestimmte Bauwerkseigenschaften, sowohl explizit auf der semantischen Ebene, als auch implizit auf Geometrieebene vorhalten. Ein Beispiel hierfür ist die Zuordnung von Bauteilen mit räumlichen Strukturen wie Stockwerken und Räumen. Des Weiteren muss bei einem 4D-Gebäudemodell die raum-zeitlich-semantische Konsistenz sichergestellt werden. Diese ergibt sich aus *technologischen Abhängigkeiten* (Braun et al., 2014). Beispielsweise kann eine tragende Stütze erst realisiert werden, wenn die darunterliegende Bodenplatte fertiggestellt wurde.

Die **Erstellung von Teilmodellen** beschreibt die Selektion einer echten Untermenge aus dem Datenbestand des Gesamtmodells. Die Selektion ist dabei für ein bestimmtes Gewerk oder die anstehenden Arbeitsschritte ausgelegt und wird zu einem eigenständigen Modell aufbereitet. Die reduzierte Datenbasis des Teilmodells unterstützt den Anwender in der effizienten Ausführung der Bearbeitung und verringert die Anforderungen an die eingesetzten Anwendungen und die Hardware.

4.2 Definition von Musteranfragen

In diesem Abschnitt werden Musteranfragen bezüglich eines IFC-Gebäudemodells definiert. Die Anfragen werden im Folgenden in den jeweils betrachteten Modellierungs- und Anfragekonzepten implementiert. In Kapitel 6.8 werden diese Musteranfragen mit der neu entwi-

ckelten BIM-Anfragesprache umgesetzt. Für die Anfragen dient die Selektion von Wänden als Ausgangspunkt. Bauteile, die in Wandaussparungen eingebettet sind, werden ebenfalls betrachtet. Außerdem wird das Enthaltensein von Wänden in der räumlichen Struktur des Gebäudes untersucht und mit einer zeitlichen Analyse kombiniert. Ein räumliche Analyse zur Untersuchung topologischer und direktionaler Relationen zwischen Wänden und Deckenplatten stellt die letzte Musteranfrage dar.

Im IFC-Modell werden Öffnungselemente (IfcOpeningElements) genutzt, um Aussparungen semantisch und geometrisch vorzuhalten. In der geometrischen Modellierung dienen sie als boolesche Subtraktionsobjekte. Abbildung 4.2 zeigt eine Wand, zwei Öffnungselemente und eine beherbergte Tür sowie deren gegenseitige Beziehungen.

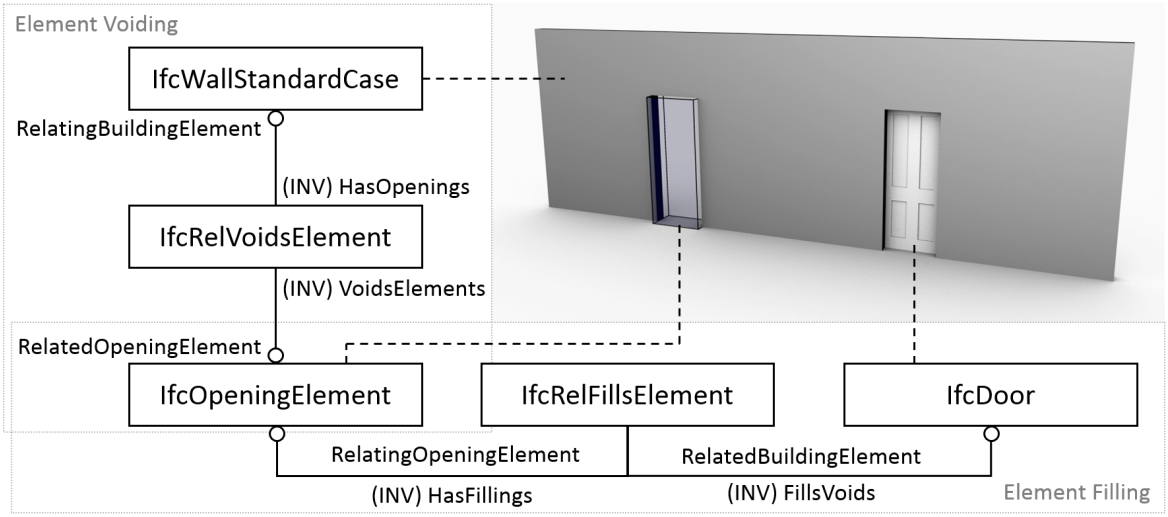


Abbildung 4.2: Relationen zwischen IfcWallStandardCase, IfcOpeningElement und IfcDoor, EXPRESS-G-Diagramm

Des Weiteren soll die räumliche Strukturierung des Gebäudes untersucht und Wände innerhalb eines bestimmten Stockwerks identifiziert werden. Im IFC-Schema wird die objektivierte Relation IfcRelContainedInSpatialStructure genutzt, um Bauteile mit einer räumlichen Struktur zu verknüpfen. Dabei kann IfcRelContainedInSpatialStructure über das Attribut *RelatedElements* mehrere Bauwerkselemente einer räumlichen Struktur zuordnen. Abbildung 4.3 zeigt das entsprechende EXPRESS-G-Diagramm.

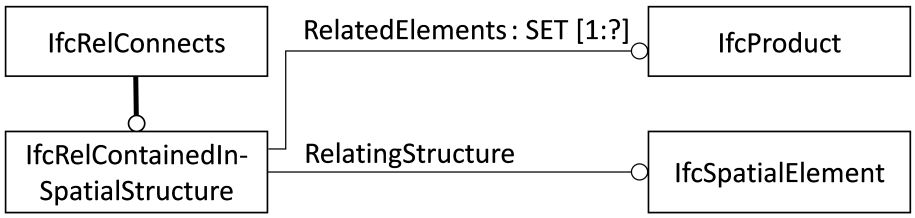


Abbildung 4.3: Assoziation von Bauelementen mit einer räumlichen Struktur über die objektivierte Relation IfcRelContainedInSpatialStructure, EXPRESS-G-Diagramm

Die Verknüpfung von Bauwerkskomponenten mit den zeitlichen Daten der Bauablaufplanung wurde bereits in Kapitel 3.5.3 besprochen (vgl. Abbildung 3.14).

Auf Basis der beschriebenen IFC-Modellierung von Wänden, beherbergten Bauteilen und Assoziationen zwischen Bauteilen und zeitlichen sowie räumlichen Strukturen können vier beispielhafte Musteranfragen in natürlicher Sprache formuliert werden:

- Musteranfrage 1 (Wände)
  „Selektiere alle Wände."
- Musteranfrage 2 (Wand/Tür-Paare)
  „Selektiere Paare aus Wand und beherbergter Tür. Dabei sollen nur Paare berücksichtigt werden, in denen die Tür mindesten 2000 mm hoch ist."
- Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerks)
  „Selektiere die Wände, die im Stockwerk mit Namen FirstFloor enthalten sind und als letztes fertiggestellt werden."
- Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand)
  „Selektiere Paare aus Deckenplatte und Wand. Dabei sollen nur Paare berücksichtigt werden in denen sich Deckenplatte und Wand berühren und sich die Wand komplett unterhalb der Deckenplatte befindet."

4.3 Analyse auf Basis relationaler Modellierung

Relationale Datenbanken stellen den Defakto-Standard in der Datenbanktechnologie dar. Konzeptionell basieren sie auf den mathematischen Grundlagen des *Relationalen Kalküls* und der *Relationalen Algebra* (Klug, 1982, Kap. 2.3 & 2.4). Das zentrale Element der Betrachtung ist hierbei die *Relation*. Von Codd (1979, Kap. 2.1) wird eine Relation wie folgt definiert:

> „Let $D_1, D_2, ..., D_n$ be $n (n > 0)$ domains (not necessarily distinct). The Cartesian product $\times\{D_i : i = 1, 2, ..., n\}$ is the set of all n-tuples $\langle t_1, t_2, ..., t_n \rangle$ such that $t_i \in D_i$ for all i. A relation R is defined on these n domains if it is a subset of this *Cartesian product*. Such a relation is said to be of degree n. In place of the index set $(1, 2, ..., n)$ we may use any unordered set, provided we associate with each tuple component not only its domain, but also its distinct index, which we shall henceforth call its attribute."

Um eine Relation darzustellen wird in dieser Arbeit folgende Syntax verwendet: Die Relation R mit *Rang* n und den *Attributen* $A_i : i = 1, 2, ..., n$ wird als $R(A_1, A_2, ..., A_n)$ notiert. Zur eindeutigen Identifikation eines bestimmten *Tupel*s der Relation dient ein *Primärschlüssel* (engl.

primary key). Dieser besteht aus einem Attribut bzw. aus mehreren Attributen der Relation, wobei im Falle eines zusammengesetzten Schlüssels nur die minimal benötigten Attribute einbezogen werden dürfen (Codd, 1979, S. 23, vergl. *uniqueness* u. *minimality property*). Beziehungen zwischen Relationen werden im relationalen Modell durch *Fremdschlüssel* (engl. *foreign key*s) ausgedrückt. Der Fremdschlüssel innerhalb einer Relation R_1 ist eine Attributsmenge, die bezüglich aller Wertebereiche mit denen eines Primärschlüssels in einer Relation R_2 übereinstimmt. Der Fremdschlüssel in R_1 referenziert eine Entität aus R_2 durch Vorhalten des entsprechenden Primärschlüsselswertes. Für eine ausführlichere Erläuterung von Primär- und Fremdschlüsseln wird auf (Elmasri et al., 2002, Kap. 7.2.2. und 7.2.4) verwiesen.

4.3.1 Operatoren der relationalen Algebra

Die relationale Algebra stellt Operatoren bereit, um Relationen zu analysieren und weiterzuverarbeiten. Das Ergebnis einer Operation ist hierbei wiederum eine Relation. Dies spiegelt die *Abgeschlossenheit* der Algebra wieder. Somit lassen sich komplexe, algebraische Ausdrücke bilden, die aus mehreren Einzeloperationen bestehen. Die Basisoperatoren der relationalen Algebra werden im Folgenden auf Basis von (Codd, 1990, Kapitel 4.2) und (Kemper, 2004, Kapitel 3.4) beschrieben.

Projektion $\pi_{A'_i,...}(R)$: Der Operator π wählt alle Tupel einer Relation R aus, übernimmt jedoch nur ausgezeichnete Attribute. A'_i muss daher ein Attribut von R darstellen.

Theta-Selektion $\sigma_\vartheta(R)$: Der Operator σ_ϑ wählt diejenigen Tupel aus, die dem *Selektionsprädikat* ϑ genügen. Die Prädikatenlogik ermöglicht die Auswertung von Tupelattributen gegenüber Konstanten und untereinander. Typische Operatoren innerhalb der *Prädikatenlogik* sind: $=, \leq, \geq, \neq$;
Beispiel eines Selektionsprädikates: $\vartheta := R.A_1 > R.A_2 \wedge R.A_1 = 5$;

Theta-Join $R \bowtie_\vartheta S$: Der Operator $\bowtie_\vartheta$ erzeugt aus den zwei Relationen R und S eine neue Relation U. Dazu wird das kartesische Produkt $R \times S$ gebildet, so dass jedes Tupel aus R mit jedem Tupel aus S vereinigt wird. Dadurch ergibt sich die neue Relation zu $U(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$. Auf U wird anschließend eine *Theta-Selektion* angewendet, in der Attribute von R und S ausgewertet werden können. Typischerweise wird die Gleichheit ausgewählter Attribute beider Eingangsrelationen überprüft. Dann wird der Operator als Equi-Join bezeichnet. Es gilt: $R \bowtie_\vartheta S = \sigma_\vartheta(R \times S)$;
Beispiel eines Theta-Join-Selektionsprädikats: $\vartheta := R.A_1 > S.B_2$;
Beispiel eines Equi-Join-Selektionsprädikats: $\vartheta := R.A_1 = S.B_2$;

Natürlicher Join $R \bowtie S$: Der Operator ist eine Erweiterung des Equi-Joins in dem identisch benannte Attribute der Relationen R und S auf Gleichheit getestet werden. Zur

Vermeidung von Redundanzen werden zudem die ausgewerteten Attribute einer Relation durch Projektion aus der Ergebnisrelation entfernt.

Relationale Vereinigung $R \cup S$: Der Operator überträgt die mengenbasierte mathematische Vereinigungsoperation auf Relationen. Dabei ist die relationale Variante nur auf die zwei Relationen R und S anwendbar, wenn sie identische Ränge aufweisen und ihre Attribute zueinander kompatibel sind (Codd, 1990, S. 79, vgl. *union compatible*). Diese Voraussetzungen müssen auch für die Verwendung der nächsten zwei Operatoren erfüllt sein.

Relationaler Schnitt $R \cap S$: Der Operator liefert eine Ergebnisrelation, die ausschließlich Tupel enthält, die sowohl in R als auch in S enthalten sind.

Relationale Differenz $R - S$: Der Operator liefert eine Ergebnisrelation, die ausschließlich Tupel von R enthält, die nicht in S enthalten sind.

Relationale Teilung $R \div S$: Damit die relationale Division zwischen R und S ausführbar ist, müssen die Attribute von S eine Teilmenge der Attribute von R darstellten. Die Ergebnisrelation U der Operation erhält alle Attribute von R, die nicht in S enthalten sind. Sind t_r und t_s Tupel aus R bzw. S muss für jedes Tupel t in U gelten:
$\forall A \in S : t_r.A = t_s.A \wedge t_r.(R - S) = t$;

Abbildung 4.4 veranschaulicht die beschriebenen Basisoperationen der Relationalen Algebra.

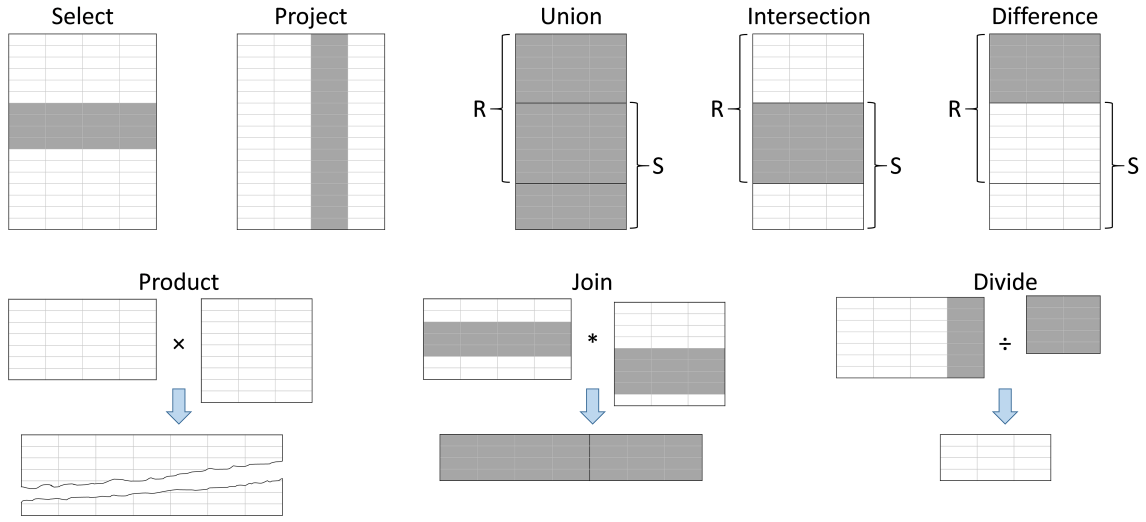


Abbildung 4.4: Die Basisoperationen der relationalen Algebra, nach (Codd, 1990, S. 78)

Die besprochenen Join-Operatoren existieren in mehreren Varianten die jeweils unterschiedlich mit nicht-verknüpfbaren Tupeln umgehen. Werden diese in der Ergebnismenge beibehalten, spricht man von einem *outer join*. Abbildung 4.5 stellt einen *inner join*, der keine unverknüpften Tupeln enthält, den drei möglichen äußeren Joins gegenüber.

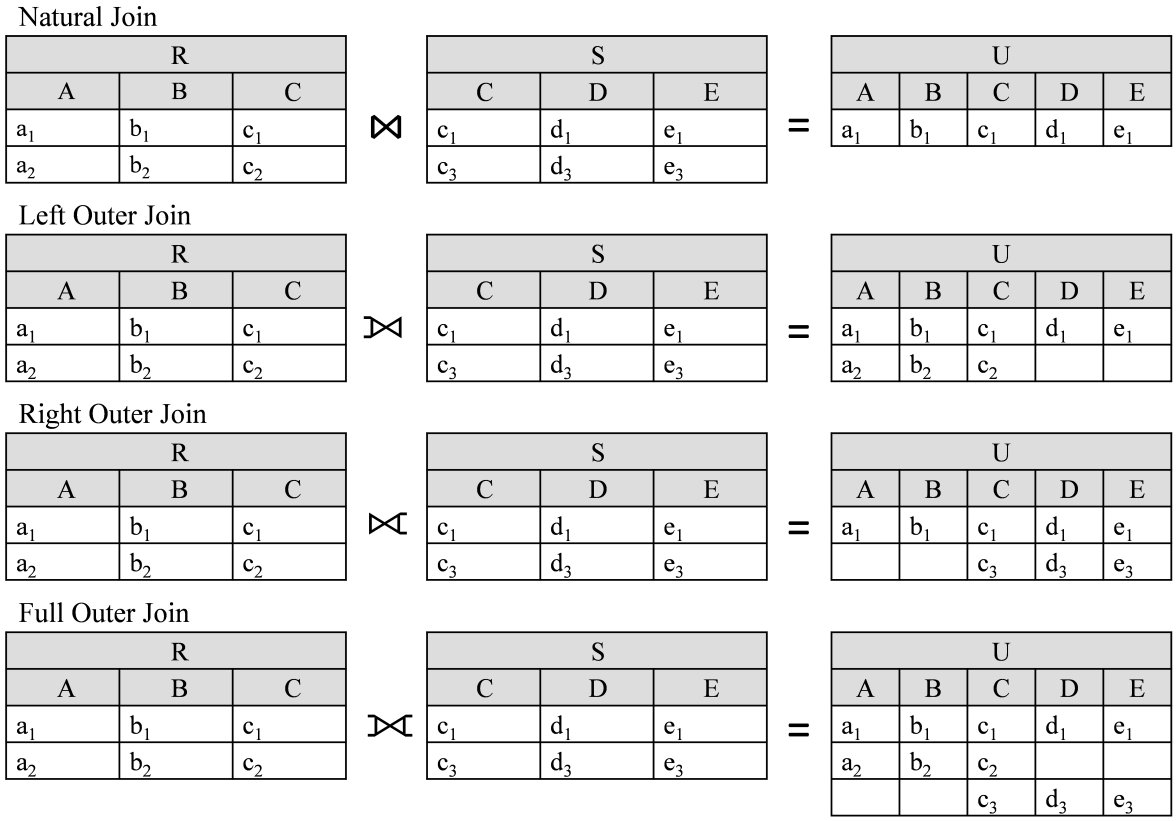


Abbildung 4.5: Natürlicher, innerer Join im Vergleich zu äußeren Joins, $\vartheta := R.C = S.C$, nach (Kemper, 2004, S. 91)

Neben diesen Basisoperatoren existieren in der relationalen Algebra auch Operatoren zur Manipulation der Daten. Dies umfasst unter anderem den `Insert-`, den `Update-` und den `Delete-`Operator. Die in dieser Arbeit entwickelte BIM-Anfragesprache ist ausschließlich als lesende Schnittstelle konzipiert. Daher werden die Manipulationsoperatoren der relationalen Algebra nicht besprochen und diesbezüglich auf (Codd, 1990, Kap. 4.3) verwiesen.

4.3.2 Transformation zum relationalen Schema

Die standardmäßige Schnittstelle relationaler Datenbanken zur Definition von Schemata, zur Datenanalyse und zur Datenmanipulation ist die *Structured Query Language (SQL)*. Diese deklarative Sprache ist seit 1989 als ISO-Standard verfügbar und wurde seitdem kontinuierlich weiterentwickelt (Islam et al., 2015). Die derzeit aktuelle Version ist *SQL:2016* (ISO, 2016b). Trotz der Normierung der Sprache nutzen relationale Datenbanken in der Regel leicht unterschiedliche *SQL-Dialekte*. Innerhalb dieser Arbeit wird zur Anfragedefinition *Transact-SQL* des *Microsoft SQL-Servers 2014* verwendet (Microsoft Developer Network, 2014).

In der objektorientierten Modellierung werden Vererbung, Kapselung und direkte Objektreferenzen eingesetzt. Das relationale Modell basiert hingegen auf normalisierten Relationen und symbolischen Referenzbeziehungen, die durch Join-Operationen aufgelöst werden müssen. Die sich hierdurch ergebende Diskrepanz zwischen objektorientierter und relationaler Modellierung wird als *Object Relational Impedance Mismatch* bezeichnet (Behm et al., 1997).

Diese Diskrepanz tritt auch bei der hier nötigen Überführungen der objektorientierten IFC-Modellierung in eine relationale Repräsentation auf. Daher müssen weitreichende Schematransformationen durchgeführt werden. In einem ersten Schritt wird eine Zuordnung von EXPRESS-Typen zu Transact-SQL-Typen aufgestellt (siehe Tabelle 4.1).

Modellierungs-ebene	Zugrundeliegender Typ/Funktion	SQL-Type Server 2014	
TYPE	NUMBER	int	a)
	REAL	float	b)
	STRING	varchar (bei IfcDate: Date)	c)
	BOOLEAN	tinyint	d)
	ENUMERATION	varchar	e)
	ARRAY, SET, LIST, BAG	zusätzliche Relation	f)
	SELECT	Datentyp aus a)-e)	g)
ENTITY	Verweis	Fremdschlüssel	h)
	ARRAY, SET, LIST, BAG	zusätzliche Relation	i)
	SELECT	Fremdschlüssel zuzüglich int	j)
	Instanz	Tupel aus allen vorherigen Elementen	k)

Tabelle 4.1: Überführung von EXPRESS-Typen zu Typen von Transact-SQL

Ein EXPRESS-Objekt wird hierbei als Tupel vorgehalten, wobei zwischen den Subtypen von TYPE und ENTITY zu unterscheiden ist. TYPE-basierte Objekte des IFC-Schemas lassen sich direkt auf SQL-Typen abbilden (a-e). Eine Besonderheit stellen dabei stringbasierte, zeitliche Werte dar (vgl. Kap. 3.5.3). Um zeitliche Auswertungen zu unterstützen, müssen diese in den SQL-Typ Date konvertiert werden. TYPEs können außerdem als Container agieren. In diesem Fall muss eine weitere Relation genutzt werden, um die enthaltenen Objekte zu referenzieren (f). SELECT-Typen dienen dazu, unterschiedliche Klassen ohne Vererbungsbeziehung einheitlich anzusprechen. Eine relationale Vorhaltung erfordert in diesem Fall neben dem Datenwert auch eine Angabe, welcher Subtyp gewählt wurde. Dies kann beispielsweise über einen Integerwert geschehen (g). Für Subklassen von ENTITY ergeben sich drei weitere Fälle. Werden einzelne Objekte referenziert, können diese Verweise über Fremdschlüssel abgebildet werden (h). Im Falle von referenzierten Containern muss wiederum eine zusätzliche Relation eingesetzt werden (i). SELECTs können ebenfalls mehrere Objekte beinhalten (j). Durch die beschriebene Abbildung lässt sich eine ENTITY, zentrales Konstrukt der EXPRESS-Modellierung, als Tupel aus Instanzen der genannten Klassen repräsentieren (k).

Auf Grund von 707 Typableitungen innerhalb des IFC4-Schemas ist die Abbildung von Vererbungsbeziehungen bei der Überführung in ein relationales Schema ein wichtiger Aspekt. In Abbildung 4.6 ist ein Ausschnitt der Vererbungshierarchie des Schemas dargestellt. Die einsetzbaren Abbildungsvarianten beeinflussen das Datenbankschema und wirken sich auf Performanz, Anfragefunktionalität und Komplexität in der Anfragedefinition aus. Scott (2000) nennt drei grundlegende Vorgehensweisen zur Abbildung einer Vererbungshierarchie in einem relationalen Modell:

1. Alle Entitäten einer Vererbungslinie werden in einer einzigen Relation abgebildet.
2. Jede nicht-abstrakte Entität wird in einer Relation vorgehalten. Alle geerbten Attribute sind in dieser Relation explizit enthalten.
3. Jeder Entität wird eine Relation zugeordnet.

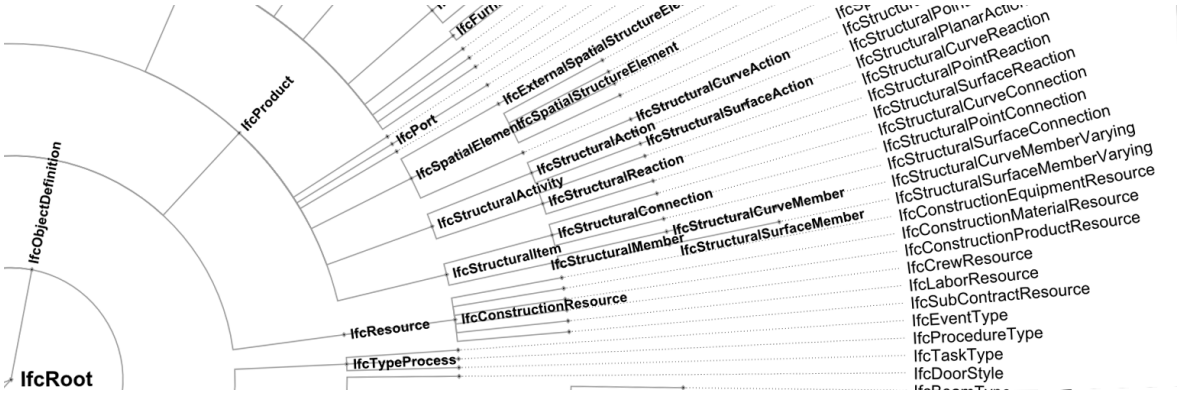


Abbildung 4.6: Ausschnitt aus der Vererbungshierarchie des IFC4-Schemas

Bei Anwendung der **ersten Modellierungsvariante** werden die unterschiedlichen Attribute aller Klassen einer Vererbungshierarchie in einer Relation zusammengeführt. Im Falle der IFC ist die Klasse IfcRoot der Ausgangspunkt dieser Aggregation. Abgesehen von Entitäten, die nicht von IfcRoot abgeleitet werden, ergibt sich eine Datenbank, die lediglich aus einer Hauptrelation besteht. Diese müsste über 300 Attribute aufnehmen, was jedoch als nicht praktikabel einzustufen ist. Außerdem würden die enthaltenen Tupel hauptsächlich mit *NULL*-Werten belegt sein und die ursprüngliche Klassenzugehörigkeit müsste als Attribut vorgehalten werden. Trotz dieser negativen Effekte kann die mehrfache Anwendung einer derartigen objekt-relationalen Abbildung auf tieferen Ebenen der IFC-Vererbungshierarchie sinnvoll sein. Anstelle von über 600 Entitätsrelationen kann dadurch ein Datenbankschema mit einem Bruchteil an Relationen realisiert werden.

Bei Anwendung der **zweiten Modellierungsvariante** werden alle instanziierbaren Klassen in das Datenbankschema aufgenommen. Eine nicht-abstrakte Superklasse erhält dabei eine eigene Relation, obwohl ihre Subklassen alle geerbten Attribute in den ebenfalls vorhandenen

expliziten Relationen wiederholen. Bei diesem Vorgehen ist somit für jede instanziierbare IFC-Klasse, wie beispielsweise IfcWall und IfcWallStandardCase, eine eigene Relation verfügbar.

Die **dritte Modellierungsvariante** überführt das komplette, objektorientierte Schema auf eine verteilte, relationale Repräsentation. Dabei werden (abstrakte) Superklassen und Subklassen jeweils getrennt auf mehrere Relationen aufgeteilt. Somit wird keine Attributsaggregation, wie in Variante 1, durchgeführt. Redundanzen zwischen Super- und Subklassen, wie sie in Variante 2 vorkommen, werden außerdem vermieden. Zur Wiederherstellung einer kompletten Entität des objektorientierten Schemas muss diese durch Equi-Joins entsprechend der Vererbungshierarchie verknüpft werden.

Um die Typhierarchie der IFC in eine relationale Repräsentation zu überführen wird die zweite Modellierungsvariante gewählt. Gründe dafür sind, dass Variante 1 in ihrer vollen Anwendung als nicht praktikabel einzustufen ist. Durch Variante 3 steigt die Komplexität von Anfragen weiter, da Entitäten auf Basis ihrer Vererbung erst durch Joins aggregiert werden müssen.

4.3.3 Musteranfragen in der Structured Query Language

In der verwendeten Modellierungsvariante 2 ergeben sich für IfcWallStandardCase, IfcRelVoidsElement und IfcOpeningElement die in Auflistung 4.1 dargestellten Relationsdefinitionen. Die Primär- und Fremdschlüsselattribute der drei Klassen sind durch Unterstreichungen gekennzeichnet, wobei das *Id*-Attribut stets als Primärschlüssel dient. Im Anhang sind zusätzlich für alle in den Musteranfragen genutzten Entitäten die entsprechenden Relationsdefinitionen enthalten (siehe Abschn. A.6).

```
IfcWallStandardCase(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar,
Description:varchar, ObjectType:varchar, ObjectPlacement:FK, Representation:
FK, Tag:varchar, PredefinedTyp:varchar)

IfcRelVoidsElement(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar,
Description:varchar, RelatingBuildingElement:FK, RelatedOpeningElement:FK)

IfcOpeningElement(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar,
Description:varchar, ObjectType:varchar, ObjectPlacement:FK, Representation:
FK, PredefinedTyp:varchar)
```

Auflistung 4.1: Relationsdefinitionen für IfcWallStandardCase, IfcRelVoidsElement und IfcOpeningElement nach Tabelle 4.1 und der zweiten Vererbungsvariante

Im Folgenden werden die Musteranfragen aus Abschnitt 4.2 als Transact-SQL-Anweisungen umgesetzt. Für eine detaillierte Syntaxbeschreibung sei diesbezüglich auf (Itzik et al., 2015) verwiesen. In Auflistung 4.2 ist Musteranfrage 1 (Wände) dargestellt.

```
SELECT * FROM IfcWallStandardCase;
```

Auflistung 4.2: Musteranfrage 1 (Wände) in Transact-SQL

Die Anfrage kann durch einen einfachen `SELECT-FROM`-Ausdruck realisiert werden. Die Typselektion erfolgt hierbei durch Auswahl der Tabelle `IfcWallStandardCase`. Einen Teil der sich ergebenden Relation zeigt Tabelle 4.2.

Id	GlobalId	Name	Tag	OwnerHistory	...
145	1XbdOf4...	Stahlbeton WU 40.0:149575	149575	41	
738	3grU21Y...	Stahlbeton WU 40.0:149555	149555	41	
Fünf zusätzliche Attribute und 2261 weitere Entitäten					

Tabelle 4.2: Teil des Ergebnisrelation von Musteranfrage 1 (Wände)

Innerhalb dieser ersten Analyse muss lediglich eine einzelne Relation der Datenbank verarbeitet werden. Dies ändert sich in den nachfolgenden Anfragen. Die SQL-Repräsentation von Musteranfrage 2 (Wand/Tür Paare) ist in Auflistungen 4.3 dargestellt.

```
1 SELECT Wall.*, Door.* FROM IfcRelVoidsElement RelV
2   JOIN IfcWallStandardCase Wall ON RelV.RelatingBuildingElement = Wall.Id
3   JOIN IfcOpeningElement Op ON RelV.RelatedOpeningElement = Op.Id
4   JOIN IfcRelFillsElement RelF ON RelF.RelatingOpeningElement = Op.Id
5   JOIN IfcDoor Door ON RelF.RelatedBuildingElement = Door.Id
6   WHERE Door.OverallHeight >= 2000;
```

Auflistung 4.3: Musteranfrage 2 (Wand/Tür-Paare) in Transact-SQL

In Anfrage 2 müssen entsprechend der in Abbildung 4.2 dargestellten Modellierung vier Join-Anfragen ausgeführt werden (Z. 2-5). In diesen werden die Entitäten durch die Auswahl der jeweiligen Primär- und Fremdschlüssel verknüpft und eine entsprechende Gesamtrelation erstellt. Diese enthält die Zuordnung von Wänden mit ihren beherbergten Türen. Durch eine Projektion werden die Attribute aus IfcRelVoidsElement, IfcOpeningElement und IfcRelFillsElement in der Ausgabe entfernt und lediglich Wand- und Türattribute ausgegeben (Z. 1). Mit Hilfe einer `WHERE`-Anweisung wird das Ergebnis zusätzlich auf Türen einer bestimmten Mindesthöhe beschränkt (Z. 6). Tabelle 4.3 zeigt ein Ausschnitt der Ergebnisrelation.

Die in Auflistung 4.4 aufgeführte Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerks) erfordert eine umfangreiche SQL-Umsetzung. Dies beruht zum einen auf SET-typisierten Attributen innerhalb der objektivierten Relationen IfcRelContainedInSpatialStructure und IfcRelAssignsToProduct. Für derartige Attribute müssen Container-Relationen erstellt werden, hier als `CSSElements` und `ATPObjects` bezeichnet (vgl. Tabelle 4.1, f). Daher werden drei `Join`-Anweisungen benötigt, um Wände mit ihrem beherbergenden Stockwerk zu

Wall.Id	Wall.Name	...	Door.Id	Door.Name	Door.OverallHeight	...
185	Trennwand		234	Single-Flush:T30 RS Tür	2260	
224	Trennwand		143	Single-Flush:T30 Tür	2735	
412	Trennwand		162	Single-Flush:T30 RS Tür	2260	
Acht bzw. elf zusätzliche Attribute und 410 weitere Paare						

Tabelle 4.3: Teil der Ergebnisrelation von Musteranfrage 2 (Wand/Tür-Paare)

verknüpfen (Z. 2-5). Weitere vier Joins sind nötig, um Wände mit ihren Realisierungszeiträumen in Verbindung zu bringen (Z. 6-9). Das `WHERE`-Prädikat schränkt das Ergebnis auf Tupel mit dem gesuchten Stockwerksnamen ein (Z. 10). Durch eine Projektion werden außerdem nur Wandattribute und der Zeitwert des *ActualFinish*-Attributs zurückgegeben (Z. 1). Die sich ergebende Relation muss nun mehrfach verarbeitet werden. Daher wird sie in einer vorher erstellten temporären Relation `#WallTime` abgelegt. Im zweiten Teil der Anfrage werden aus der temporären Relation diejenigen Wände selektiert, die mit dem maximalen Wert im *ActualFinish*-Attribut assoziiert sind (Z. 12 u. 13).

```
1  SELECT Wall.*, Time.ActualFinish INTO #WallTime
2  FROM IfcRelCSS RelCss
3  JOIN CSSElements RelE ON RelCss.ElementsSetId = RelE.SetId
4  JOIN IfcWallStandardCase Wall ON RelE.RelatedElementId = Wall.Id
5  JOIN IfcBuildingStorey Sto ON RelCss.RelatingStructure = Sto.Id
6  JOIN IfcRelAssignsToProduct RelATP ON RelATP.RelatingProduct = Wall.Id
7  JOIN ATPObjects RelO ON RelATP.RelatedObjectsSetId = RelO.SetId
8  JOIN IfcTask Task ON RelO.RelatedObjectId = Task.Id
9  JOIN IfcTaskTime Time ON Task.TaskTimeId = Time.Id
10 WHERE Sto.Name = 'FirstFloor'
11
12 SELECT * FROM #WallTime
13 WHERE ActualFinish = (SELECT MAX(ActualFinish) FROM #WallTime)
```

Auflistung 4.4: Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) in Transact-SQL

Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) lässt sich in Standard-SQL nicht umsetzten. Auch in räumlichen SQL, das zumindest topologische Operatoren für zweidimensionale Geometrierepräsentationen bereitstellt, fehlen Möglichkeiten für eine direktionale Auswertung.

Die Analyse von Gebäudemodellen auf Basis von relationalen Datenbanken und SQL gestaltet sich als komplex. Neben der zugrundeliegenden relationalen Modellierung müssen die Eigenschaften der eingesetzten Transformation berücksichtigt werden. Aus dem relationalen Modell ergibt sich bei der Auswertung von Objektverweisen die Notwendigkeit aufwendige Join-Anweisungen einzusetzen. Außerdem erfordern Container-

und SELECT-Typen zusätzliche Relationen bzw. Attribute. Dies erschwert nochmals die Anfragedefinition für Fachanwender. Die Analyse von IFC-Vererbungsbeziehungen wird im relationalen Modell nicht direkt unterstützt.

4.4 Objektorientierte Analyse

Seit den achtziger Jahren des letzten Jahrhunderts hat die zunehmende Objektorientierung die Implementierung und den Entwurf von Softwareprodukten geprägt. Objektorientierte (OO) Konzepte werden zum einen als Programmierparadigma in *Hochsprachen* wie *C++*, *C#* und *Java* genutzt und kommen bei der Erstellung von Datenmodellen wie den Industry Foundation Classes und CityGML zum Einsatz. Zudem wurden OO-Konzepte in Datenbanken integriert, was zur Entwicklung von *Objektdatenbanken* und *objekt-relationalen* Datenbanken geführt hat. Hauptziele für die Entwicklung und Nutzung der Objektorientierung sind die Effizienzsteigerung in der Softwareentwicklung sowie eine hohe Implementierungsqualität und eine bessere Wartbarkeit der zu realisierenden Anwendungen (Schiffer, 1998, S. 115). Im Folgenden wird auf objektorientierte Ansätze zur Analyse von IFC-Daten eingegangen.

4.4.1 Object Query Language

Die unterschiedlichen Ansätze in der objektorientierten und der relationalen Modellierung führen zum bereits genannten Object Relational Impedance Mismatch. Wie im letzten Abschnitt am Beispiel der IFC gezeigt, tritt dieses beim Zusammentreffen von OO-Modellierung bzw. -Programmierung und relationaler Datenhaltung auf. Um die verwendeten Ansätze anzugleichen wurden Objektdatenbanken entwickelt.

Erste Bestrebungen für eine Standardisierung von Objektdatenbanken gingen von der *Object Database Management Group (ODMG)*, einem Zusammenschluss mehrerer Softwarehersteller, aus (Zicari, 2000). Die ODMG veröffentlichte dazu zwischen 1993 und 2001 die ODMG-Spezifikation, zuletzt in der Version 3.0 (Cattell, 2000). Diese beinhaltet mit der *Object Query Language (OQL)* eine Anfragesprache. OQL basiert auf SQL:92 und wurde mit Funktionen zur Analyse objektorientierter Datenmodelle erweitert.

Anstelle des Konzeptes der Primär- und Fremdschlüssel kann in ODMG-konformen Systemen mit direkten Objektverweisen gearbeitet werden. Dies erleichtert das Traversieren von Objektbeziehungen in OQL-Anfragen, da auf Join-Operationen verzichtet werden kann. Um Referenzen aufzulösen kann stattdessen der *Punkt*-Operator genutzt werde. Eine mehrmalige Anwendung des Punkt-Operators ist jedoch nur bei 1:1-Beziehungen möglich.

Aufzählung 4.5 zeigt die Umsetzung von Musteranfrage 2 (Wand/Tür Paare) in OQL. Zu Beginn der `SELECT`-Anweisung erfolgt, wie in standardmäßigen SQL, die Auswahl der zurückgegebenen Elemente, hier `Wall` und `Door` (Z. 1). Durch wiederholte Anwendung des Punkt-Operators kann von IfcRelVoidsElement zu beherbergenden Elementen und zu ihren Aussparungsobjekten (IfcOpeningElement) traversiert werden. Durch die Anwendung des inversen Attributes *HasFillings* gelangt man vom Aussparungsobjekt zu IfcRelFillsElement und schließlich zum beherbergten Element. Durch drei Attributsprüfungen innerhalb des `WHERE`-Prädikats erfolgt die Reduktion auf Wände und Türen der geforderten Höhe (Z. 5 u. 6).

```
1 SELECT Wall, Door FROM IfcRelVoidsElement RelV,
2   RelV.RelatingBuildingElement Wall,
3   RelV.RelatedOpeningElement.HasFillings RelF,
4   RelF.RelatedBuildingElement Door
5 WHERE Wall.IfcType = "IfcWallStandardCase" AND Door.IfcType = "IfcDoor"
6   AND Door.OverallHeight >= 2000;
```

Auflistung 4.5: Musteranfrage 2 (Wand/Tür-Paare) in OQL 3.0

Die direkte Traversierung von Objektreferenzen in OQL erleichtert die Definition von Anfragen, insbesondere wenn 1:1-Relationen zwischen den Entitäten bestehen. Dabei zeigt sich die Nutzung von inversen Attributen des feingranularen IFC-Objektmodells als vorteilhaft, da hierdurch die Richtung der Verweisauflösung beibehalten werden kann. Die fehlende Analysefunktion für Vererbungsbeziehungen ist jedoch auch in OQL-basierten Anfragen gegeben. Außerdem ist die Verfügbarkeit von ODMG 3.0-konformen Systemen stark eingeschränkt (Apache Software Foundation, 2011).

4.4.2 Die Mapping-Sprache EXPRESS-X

Der Standard for the Exchange of Product model data (STEP) beinhaltet keine reine Anfragesprache. Mit *EXPRESS-X* wurde jedoch ein deklarative *Mapping Language* integriert. Die Sprache dient dazu Entitäten eines EXPRESS-Schemas auf die Entitäten eines anderen Schemas abzubilden. Somit können EXPRESS-basierte Datenmodelle für die jeweiligen Anwendungen aufbereitet werden (Bailey et al., 1996). Die Abbildung von Entitäten unterschiedlicher Schemata wird über sogenannte *Views* realisiert. Diese lassen sich auch für die Datenfilterung nutzen, wodurch sich die Mapping-Sprache wie eine Anfragesprache verhält. In der Anfragedefinition fallen Ähnlichkeiten zur SQL und OQL auf.

Aufzählung 4.6 zeigt die Umsetzung von Musteranfrage 2 (Wand/Tür Paare) in EXPRESS-X. Zu Beginn der Anfrage muss ein `VIEW` definiert werden, in der die nachfolgenden Anweisungen eingebettet werden (Z. 1). Die Auswahl der Datenquelle, hier eine Enumeration aus `IfcWallStandardCase`-Entitäten, erfolgt durch die `FROM`-Anweisung in Zeile 2. In der nach-

folgenden `WHERE`-Anweisung wird durch wiederholte Anwendung des Punkt-Operators zu den beherbergten Elementen traversiert (Z. 3). Bei containerbasierten Attributen muss dabei ein expliziter Index angegeben werden. Daher wird in der Anfrage die erste Öffnung und das erste beherbergte Element zurückgegeben. Um Musteranfrage 2 in ihrer ursprünglichen Form zu realisieren, müssten Join-Operatoren in der Anfrage angewendet werden.

```
1 VIEW sample2
2   FROM (w : IFCWallStandardCase);
3   WHERE (w.HasOpenings[1].RelatedOpeningElement.HasFillings[1].
    RelatedBuildingElement.OverallHeight >= 2000);
4 END VIEW;
```

Auflistung 4.6: Teilumsetzung von Musteranfrage 2 (Wand/Tür-Paare) in EXPRESS-X

Auch in EXPRESS-X erleichtert die direkte Traversierung von Objektreferenzen die Definition von Anfragen. Jedoch besteht keine Möglichkeit 1:N-Relationen mit dem Punkt-Operator und ohne die Verwendung expliziter Indizes zu dereferenzieren. Daher müssen 1:N-Relationen auch in OQL mit Join-Operatoren aufgelöst werden, wodurch sich die Anfragedefinition nicht wesentlichen von der SQL-Variante unterscheidet. Die nötige Einbettung in ein Mapping-Schema erschwert zudem die Anwendung von EXPRESS-X zur Analyse von BIM-Daten (Tauscher & Smarsly, 2016, S. 2).

4.4.3 Benutzerdefinierte Datentypen in SQL

Auch das standardisierte SQL wurde mit Konzepten aus der Objektorientierung erweitert. Dazu gehören seit SQL:1999 *benutzerdefinierte Datentypen*, engl. *User Defined Types (UDTs)*. Mit UDTs entfällt die Beschränkung, dass Relationen ausschließlich aus atomaren Datentypen aufgebaut sein müssen. Außerdem können diese Datentypen Methoden veröffentlichen und somit den Funktionsumfang der Datenbank erweitern.

Von Borrmann (2007, Kapitel 9.2.4) wird die Anwendung einer objekt-relationalen Datenbank und einer SQL-basierten Anfragesprache als Grundlage für räumliche Analysen von Gebäudemodellen vorgestellt. Dazu werden benutzerdefinierte Typen innerhalb der Datenbank mit räumlichen Methoden versehen. Auflistung 4.7 zeigt eine Anfrage, in der die metrische Methode `closerThan` eines benutzerdefinierten Wand-Typs in einer `WHERE`-Anweisung genutzt wird.

```
SELECT w2.uid
FROM Wall w1, Wall w2
WHERE w2.closerThan(w1, 10.0) AND w1.uid = HJDSA32asdfh
```

Auflistung 4.7: Beispiel für eine räumliche Anfrage auf Basis von UDTs, aus (Borrmann, 2007)

Benutzerdefinierte Typen bieten eine Möglichkeit domänenspezifische Funktionen in SQL zu integrieren. Auf Basis geeigneter Geometriealgorithmen kann dadurch beispielsweise eine räumliche Analyse ermöglicht werden. Benutzerdefinierte Typen bieten jedoch keinen generellen Ansatz, um die zuvor aufgeführten, konzeptionellen Defizite einer SQL-basierten BIM-Analyse zu beseitigen.

4.4.4 Language Integrated Query

Eine eingebettete, objektorientierte Anfragesprache ist die *Language Integrated Query (LINQ)*. Sie ist Bestandteil des *.NET-Frameworks* und für die Sprachen *Visual Basic* und *C#* verfügbar (Meijer et al., 2006). Primär wurde die Anfragefunktionalität für die Filterung und Verarbeitung von Objektsammlungen wie Arrays und Listen entwickelt, kann in der jetzigen .NET-Version 4.6 jedoch auch für die Prozessierung relationaler und XML-kodierter Daten verwendet werden. Eine LINQ-Anfrage kann in zwei Syntaxvarianten erstellt werden. Die erste wird als *Fluent Syntax* bezeichnet und ähnelt einer SQL-Anweisung (Albahari & Albahari, 2012, S. 313). Die zweite Notation ist die *Query Expression Syntax (QES)*. Da die Fluent Syntax nicht alle LINQ-Operatoren bereitstellt, wird in dieser Arbeit ausschließlich die QES verwendet.

In der Anfragedefinition basiert LINQ auf einem *deklarativen* Ansatz wobei sich Anfragen in den *imperativen* Code der .NET Sprachen einbetten lassen. In der QES-Notation werden dabei *iterierbare Container* (engl. *enumerations*) durch LINQ-Operatoren gefiltert und weiterverarbeitet. Technisch basiert LINQ auf *Erweiterungsmethoden* (engl. *extension methods*), einer Möglichkeit des .NET-Frameworks bereits existierende Typen mit Methoden zu erweitern ohne diese *Rekompilieren* zu müssen. Somit sind LINQ-Operatoren wie `Where()` und `Select()` direkt an Enumerationen angebunden und lassen sich durch den Punkt-Operator wie standardmäßige Objektmethoden aufrufen. Eine vollständige Beschreibung aller LINQ-Operatoren ist in (microsoft, 2015) enthalten. Die nachfolgende Anfrage in Auflistung 4.8 zeigt die beispielhafte Filterung und Bearbeitung einer Enumeration aus Zeichenketten.

```
string[] floorMaterials = {"Parkett", "Click-Parkett", "Laminat", "Rohputz"};

IEnumerable<string> filteredFloorMaterials = floorMaterials.Where (n => n.
Contains ("Parkett")).Select(n => n.ToUpper());
```

Auflistung 4.8: Filterung und Weiterverarbeitung einer Enumeration durch eine LINQ-Anfrage, QES-Notation

Als Datenquelle wird das Array `floorMaterials` erstellt, das fünf Zeichenketten beinhaltet. Die `Where`-Methode selektiert nur die Zeichenketten, die den Substring „Parkett“ enthalten.

In der `Select`-Methode wird die Rückgabe der Anfrage festgelegt. Bei der Iteration von komplexen Typen kann an dieser Stelle durch Auswahl eines Objektattributes eine Projektion ausgeführt werden. Im aktuellen Beispiel wird der Aufruf einer im String-Typ enthaltenen Methode demonstriert. Durch `ToUpper`() werden alle enthaltenen Zeichen in Großbuchstaben umgewandelt und eine neue, veränderte Zeichenkette zurückgegeben. Der Rückgabewert `filteredFloorMaterials` ist eine Enumeration von Zeichenketten und spiegelt die starke Typisierung von LINQ wider.

Betrachtet man die Aufrufe der `Where`- und `Select`-Operatoren in Anfrage 4.8, erkennt man, dass Funktionen als Parameter übergeben werden. Dieses Konzept der LINQ-Sprache ermöglicht das Verhalten der angebotenen Operatoren genau zu steuern. Hierzu unterstützt das .NET-Framework ab Version 3.0 sogenannte *Lambda*-Ausdrücke. Diese Art der Funktionsdefinition stammt ursprünglich aus dem Bereich der *funktionalen* Programmiersprachen (Peyton Jones & Wadler, 1993). Durch einen Lambda-Ausdruck wird eine *anonyme Methode* mit *formalen Parametern* erstellt und diese implizit einem *Delegaten* zugewiesen, so dass die Funktion als Objektinstanz vorliegt. Funktionen, die als Parameter dienen, können so objektorientiert und durch eine kompakte Syntax erstellt werden. Ein Lambda-Ausdruck wird in LINQ durch den `=>`-Operator definiert, der die links stehenden, formalen Parameter mit der rechts stehenden Funktion in Verbindung setzt.

4.4.5 Musteranfragen in der Language Integrated Query

Zur Analyse von IFC-Daten über LINQ muss eine formale Abbildung des IFC-EXPRESS-Schemas bzw. des IFC-XML-Schemas auf C#-Klassen erfolgen. Dies kann durch *parsen* eines der beiden Schemata und einer entsprechenden Codegenerierung automatisch geschehen. Für die nachfolgenden Beispiele wurden IFC-Entitäten jedoch auf vereinfachte C#-Klassen abgebildet. Diese sind Teil einer reduzierten Vererbungshierarchie und beinhalten ausschließlich die benötigten Attribute. Die im Folgenden erörterten Musteranfragen werden durch diese Vereinfachung nicht beeinflusst. Die genutzten Klassendefinitionen sind im Anhang enthalten (siehe Abschn. A.7).

In Auflistung 4.9 ist Musteranfrage 1 (Wände) als LINQ-Anweisung umgesetzt. In dieser wird die `elements`-Enumeration auf IfcWallStandardCase-Instanzen reduziert. Innerhalb des Lambda-Ausdrucks wird dafür der C#-Operator `is` eingesetzt, der die Typzugehörigkeit einer Instanz prüft. Das Ergebnisse der LINQ-Anfrage stellt wiederum eine Enumeration dar und wird der Variable `walls` zugewiesen.

```
static void Main(string[] args)
{
    //the variable elements contains all IfcElements entities of the model
    var walls = elements.Where(e => e is IfcWallStandardCase);
}
```

Auflistung 4.9: Musteranfragen 1 (Wände) in LINQ

In Auflistung 4.10 wird Musteranfrage 2 (Wand/Tür-Paare) realisiert. Aus Gründen der Übersichtlichkeit werden drei sequenzielle LINQ-Anweisungen genutzt, um die Anfrage zu realisieren. Durch die Aufteilung können in nachfolgenden Anweisungen Zwischenergebnisse genutzt werden. Grundsätzlich wäre die Anfrage auch in einem *Statement* umsetzbar. Auf die Darstellung der übergeordneten C#-Funktion wird ab hier verzichtet.

Im ersten Anfrageteil werden die IfcRelVoidsElement-Relationselemente durch ein `Where`-Prädikat aus der `elements`-Enumeration ausgewählt und *gecastet* (Z. 2). Durch das nachfolgende `Where`-Prädikat wird das Ergebnis auf die Instanzen eingeschränkt, die über das *RelatingBuildingElement*-Attribut Objekte vom Typ IfcWallStandardCase referenzieren (Z. 3). Die Einschränkung auf Türen einer Mindesthöhe gestaltet sich aufwendiger (Z. 4 - 6). Zwar kann nach Selektion und casten der IfcRelFillsElement-Instanzen auch hier der `is`-Operator angewendet werden. Um auf Attribute von IfcDoor zuzugreifen, muss jedoch nochmals eine Cast-Operation erfolgen. Dies ist nötig, da im *RelatedBuildingElement*-Attribut der IfcRelFillsElement-Klasse eine Superklasse für Bauteile genutzt wird (IfcElement). Wie beschrieben, sind im IFC-Datenmodell Bauteile mit ihren Durchbrüchen assoziiert und Durchbrüche wiederum mit den sie füllenden Komponenten. Somit kann über die Durchbrüche eine Gesamtrelation aus den zwei Teilrelationen erstellt werden (Z. 7 - 10). Die dazu verwendete Join-Anweisung in LINQ verknüpft über identische Attributwerte zwei Enumerationen. Als Rückgabe wird eine Enumeration aus Tupeln erstellt. Dabei beinhaltet jedes Tupel eine Wand und die beherbergte, mindestens 2000 mm hohe Tür.

```
1  //using relVo = IfcRelVoidsElement; using relFi = IfcRelFillsElement;
2  var relV = elements.Where(e => e is relVo).Select(e => e (relVo)e)
3              .Where(v => v.RelatingBuildingElement is IfcWallStandardCase);
4  var relF = elements.Where(e => e is relFi).Select(e => e (relFi)e)
5              .Where(f => f.RelatedBuildingElement is IfcDoor &&
6              ((IfcDoor)f.RelatedBuildingElement).OverallHeight >= 2000);
7  var result = relV.Join(relF,
8              f => f.RelatedOpeningElement, v => v.RelatingOpeningElement,
9              (f, v) => new Tuple<IfcWall, IfcDoor>
10             (f.RelatingBuildingElement, v.RelatedBuildingElement);
```

Auflistung 4.10: Musteranfrage 2 (Wand/Tür Paare) in LINQ

In Auflistung 4.11 wird Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerks) durch eine Reihe an LINQ-Anfragen umgesetzt. Als erstes werden aus der übergebenen IfcRelationship-Enumeration alle IfcRelContainedInSpatialStructure-Instanzen selektiert. Um auf die Attribute dieser Subklasse zugreifen zu können, erfolgt außerdem eine Cast-Operation (Z. 3). Über IfcRelContainedInSpatialStructure werden nun alle räumlichen Strukturen mit dem Namen `FirstFloor` und deren referenzierte Bauteile gesucht. Falls es sich hierbei um Wände handelt, werden sie in der Variable `walls` abgelegt (Z. 4 - 6). Als nächstes werden IfcRelAssignsToProduct-Instanzen selektiert und eine verschachtelte Select-Anweisung genutzt, um die 1:n-Relation des *RelatedObjects*-Attributs aufzulösen. Da in LINQ Relationen nicht direkt verarbeitet werden können, muss hier einen Enumeration aus Tupeln erstellt werden. Jedes Tupel enthält eine IfcWall und ein IfcTaskTime (Z. 9 - 12). Das Ergebnis wird der Variable `wallTime` zugeordnet. Aus dieser Enumeration werden nun die Tupel ausgewählt, deren Wand-Instanz in der `walls`-Enumeration, die aus Wänden innerhalb des ersten Stockwerks besteht, enthalten ist (Z. 14). Aus den so reduzierten IfcWall/IfcTaskTime-Tupeln wird der maximale Zeitwert ermittelt und alle Tupel ausgewählt, die diesen Wert aufweisen (Z. 15 und 16).

```
1  //using relCSS = IfcRelContainedInSpatialStructure;
2  //using relAP = IfcRelAssignsToProduct;
3  var relC = relations.Where(e => e is relCSS).Select(e => e (relCSS)e));
4  var walls = relC.Where(c => c.RelatingStructure.Name == "FirstFloor")
5                .SelectMany(c => c.RelatedElements).Where(e => e is IfcWall)
6                .Select(e => (IfcWall)e);
7
8  var relA= relations.Where(e => e is relAP).Select(e => e (relAP)e));
9  var wallTime = relA.Select(r => r.RelatedObjects
10               .Where(o => r.RelatingProduct is IfcTask && o is IfcWall)
11               .Select(o => new Tuple<IfcWall, IfcTaskTime>(
12                 (IfcWall)o, ((IfcTask)r.RelatingProduct).TaskTime));
13
14 wallTime = wallTime.Where(t => walls.Contain(t.Item1));
15 var maxDate = wallTime.Max(t => t.Item2);
16 var result = wallTime.Where(t => t.Item2 == maxDate);
```

Auflistung 4.11: Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerks) in LINQ

Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) lässt sich in LINQ nur umsetzen, wenn die entsprechenden räumlichen Prädikate durch Methoden auswertbar sind (vgl. Kapitel 5.2).

LINQ erweist sich als umfassendes Konzept zur Analyse objektorientierter Datenmodelle. Die Anfragesprache ist jedoch primär als Werkzeug für Anwendungsentwickler

konzipiert. So werden Anfragen von Endnutzern, die erst zur Programmlaufzeit definiert werden, standardmäßig nicht unterstützt. Um diese Einschränkung zu umgehen, bedarf es Erweiterungen, die in Daum & Borrmann (2013) beschrieben werden.

Durch die Nutzung von Lambda-Ausdrücken als Eingangsparameter kann eine hohe Flexibilität der LINQ-Operatoren erreicht werden. Zudem können in Anfragen direkt Objektmethoden aufgerufen werden. Dies erlaubt, domänenspezifische Erweiterungen in die Anfragesprache zu integrieren. Für Fachanwender ist eine LINQ-basierte Analyse jedoch auf Grund des komplexen .NET-Typsystems und der syntaktisch anspruchsvollen Lambda-Ausdrücken problematisch. Zudem fallen die häufig benötigten Cast-Operationen auf.

Daraus kann der Schluss gezogen werden, dass in einer Anfragesprache für IFC-Daten nicht nur die Unterstützung einer Vererbungshierarchie entscheidend ist. Gleichzeitig müssen entsprechende Sprachfeature die unkomplizierte Auswertung dieser Vererbungshierarchie ermöglichen.

4.5 Analyse XML-basierter Gebäudeinformationsmodelle

Mit der Einführung der *Extensible Markup Language (XML)* und der Entwicklung abgeleiteter Auszeichnungssprachen hat die Bedeutung von *Markup*-basierten Konzepten zugenommen. So löste eine Datenmodellierung mit XML in vielen Bereichen proprietäre Ansätze ab (Cover, 2005). In der Bau- und Geoinformatik werden neben den beschriebenen Austauschformaten für Stadt- und Gebäudemodelle, MVDs und Property Sets sowie eine Reihe von OGC-Standards mit XML umgesetzt (BuildingSMART, 2017b; OGC, 2016).

Im Folgenden wird auf die Datenmodellierung mit XML eingegangen. Danach erfolgt eine Betrachtung der Anfragesprache *XQuery*, die für die Analyse von XML-Daten entwickelt wurde und somit auch ifcXML-Daten verarbeiten kann.

4.5.1 Grundlagen einer XML-basierte Modellierung

Von Fischer & Hofer (2011, S. 547) wird Markup wie folgt definiert:

> „Zeichen innerhalb eines Zeichenstroms mit nicht inhaltsbezogener, sondern mit auszeichnender Semantik oder einer Semantik auf Meta-Ebene."

XML ermöglicht Daten strukturiert, selbstbeschreibend und plattformunabhängig in Textform vorzuhalten und zu transferieren. Im Gegensatz zur *HyperText Markup Language (HTML)*

wird die Interpretation der Daten der verarbeitenden Applikation überlassen und es werden ausschließlich syntaktisch korrekte Dokumente verarbeitet (Oxford Brookes University, 2002, vgl. *well-formed documents*).

Durch die *XML Schema Definition Language (XSD)* ist es möglich, die Struktur von XML-Dokumenten für eine bestimmte Anwendung festzulegen. Dadurch lassen sich domänenspezifische XML-Anwendungen wie ifcXML, CityGML und GML realisieren sowie entsprechende Instanzdateien bezüglich ihrer Schemadefinition validieren (BuildingSMART, 2014c; OGC, 2012; Simon et al., 2004). In Auflistung 4.12 ist ein Instanzdokument eines minimalistischen XML-Dialekts für Gebäudemodelle dargestellt. Das entsprechende XSD-Schema ist im Anhang aufgeführt (siehe Abschn. A.8).

```
1 <building>
2     <wall id="274">Leichte Trennwand</wall>
3     <wall id="845">Trennwand</wall>
4     <door id="111">T30 Tür</door>
5 </building>
```

Auflistung 4.12: Instanzdokument eines minimalistischen XML-Dialekts für Gebäudemodelle

Jedes *datenzentrische* XML-Dokument beinhaltet eine Baumstruktur aus Elementen. Diese werden durch Start- und End-*Tags* begrenzt, beispielsweise `<building>` und `</building>`. Start-Tags können außerdem Attribute enthalten (siehe `Id`-Attribut innerhalb des `wall`-Tags). Für eine detaillierte Einführung in die Auszeichnungssprache und eine Beschreibung der XML-Syntax sei auf (Harold et al., 2004, Kap. 2) verwiesen.

4.5.2 Grundlagen der XQuery-Anfragesprache

Für die Filterung, Analyse und Weiterverarbeitung von XML-kodierten Inhalten hat sich XQuery, eine vom *World Wide Web Consortium (W3C)* standardisierte Anfragesprache etabliert. Sie kann sowohl für XML-Dokumente als auch in XML-Datenbanken genutzt werden. Die aktuellste Version 3.0 liegt seit April 2014 als W3C-*Recommendation* vor (W3C, 2014d).

Auf die Sprache *Quilt* zurückgehend, nutzt XQuery eine Syntax ohne Auszeichnungselemente (Walmsley, 2007, S. 29). XQuery ist primär eine deklarative Sprache, beinhaltet aber auch *prozedurale* Konzepte wie Funktionsaufrufe. Als Ausführungsumgebung wird ein *XQuery-Prozessor* genutzt, der Anfragen parst, auf Syntaxfehler überprüft und ausführt. Zur Anfrageoptimierung formt der Prozessor gegebenenfalls die Anfrage zu einer gleichwertigen, aber effizienter ausführbaren Anfrage um (Florescu et al., 2003, Kap. 7.5). Der Prozessor überführt außerdem die zu analysierenden XML-Inhalte in das für die Weiterverarbeitung geeignetere *XQuery and XPath Datamodel (XDM)* (Anders et al., 2010).

XQuery ist eine Erweiterung zur *XPath*-Sprache, die eine Syntax zur Erstellung von Pfadausdrücken bereitstellt (Berglund et al., 2007). Auch XPath nutzt für die Anfrageverarbeitung XDM als Grundlage. Pfadausdrücke in XPath dienen dazu, die aus dem jeweiligen XML-Inhalt erstellte XDM-Baumstruktur zu traversieren. Dabei können *Subbäume* (engl. *forests*), *Baumknoten* (engl. *nodes*) oder atomare Werte selektiert werden. Ein XPath-Ausdruck ist gleichzeitig auch ein valider XQuery-Ausdruck und liefert im allgemeinen das gleiche Ergebnis in beiden Ablaufumgebungen. Eine Ausnahme ist das abweichende Verhalten von XPath und XQuery in Bezug auf maskierte Sonderzeichen wie `&lt`.

Während XPath ausschließlich eine pfadbasierte Selektion aus einem XML-Datenbestand ermöglicht, ist mit XQuery eine umfangreichere Analyse und Prozessierung möglich. So können Assoziationen zwischen Objekten untersucht und Funktionen zur Weiterverarbeitung von XML-Elementen eingesetzt werden. Wie bereits erwähnt, geschieht die Typisierung von Elementen eines XML-Dokuments über die XML Schema Definition Language. Auf ein entsprechendes Schema kann in XQuery verwiesen werden (Gao et al., 2012). Diese hat den Vorteil, dass Typfehler in Anfragen mit höherer Wahrscheinlichkeit bereits zur *Kompilierzeit* gefunden werden können.

Die Syntax von XQuery 3.0 ist in der Extended Backus-Naur Form (EBNF) definiert und umfasst 196 Produktionen (W3C, 2014d, Kap. A.1). Im Folgenden wird auf zentrale Produktionen näher eingegangen. Diese sind in Auflistung 4.13 dargestellt.

```
[38]  QueryBody ::= Expr
[39]  Expr ::= ExprSingle ("," ExprSingle)*
[40]  ExprSingle ::= FLWORExpr | QuantifiedExpr | SwitchExpr  |
        TypeswitchExpr  | IfExpr  | TryCatchExpr | OrExpr
[41]  FLWORExpr ::= InitialClause IntermediateClause* ReturnClause
[42]  InitialClause ::= ForClause | LetClause | WindowClause
[43]  IntermediateClause ::= InitialClause | WhereClause | GroupByClause |
        OrderByClause | CountClause
[44]  ForClause ::= "for" ForBinding ("," ForBinding)*
[48]  LetClause ::= "let" LetBinding ("," LetBinding)*
[60]  WhereClause ::= "where" ExprSingle
[65]  OrderByClause ::= (("order" "by") | ("stable" "order" "by"))
        OrderSpecList
[69]  ReturnClause ::= "return" ExprSingle
```

Auflistung 4.13: Ausschnitt aus der XQuery Sprachdefinition, aus (W3C, 2014d, Kap. A.1)

Der zentrale Bestandteil einer XQuery-Anfrage ist der `QueryBody`, der eine oder mehrere einfache Ausdrücke (`ExprSingle`) beinhaltet (siehe Produktion 38). Vor der eigentlichen Anfrage, dementsprechend vor dem QueryBody, kann optional ein *Prolog* aufgeführt werden. In diesem können unter anderem *Namensräume*, Variablen und Funktionen deklariert sowie Schemata und Module importiert werden. Durch die `ExprSingle`-Produktion ist ersichtlich,

dass XQuery übliche Sprachkonstrukte wie konditionale Ausdrücke, Verzweigungen und Fehlerbehandlungen innerhalb eines *TryCatch*-Blocks unterstützt (Produktionen 39 u. 40). Der *FLWOR*-Ausdruck (`FLWORExpr`, Produktion 41) ist dagegen eine Besonderheit von XQuery und setzt sich aus fünf Unterausdrücken zusammen:

- **F**or bewirkt eine Iteration über eine oder mehrere Sequenzen (Produktion 44)
- **L**et deklariert eine Variable (Produktion 48)
- **W**here nimmt zur Filterung einer Eingangssequenz ein Prädikat auf (Produktion 60)
- **O**rder by dient zur Sortierung der Ergebnissequenz (Produktion 65)
- **R**eturn ermöglicht die Strukturierung der Ergebnissequenz (Produktion 69)

Die Verwendung eines FLWOR-Ausdruck wird anhand der Beispielanfrage in Auflistung 4.14 veranschaulicht.

```
1 for $item1 in ("Parkett","Click-Parkett","Laminat","Diele"),
2     $item2 in ("Variante 1","Variante 2")
3 let $upperString := upper-case($item1)
4 where contains( $item1, "Parkett")
5 order by $item1
6 return <item  first="{$item1}" second="{$item2}">{$upperString}</item>
```

Auflistung 4.14: Beispiel einer FLWOR-Anfrage in XQuery

Das Beispiel zeigt, dass mehrere Sequenzen in einem For-Ausdruck iteriert werden können. Dies führt zur Erstellung des *kartesischen Produkts* aus beiden Sequenzen (Z. 1 u. 2). In Zeile 3 wird die Variable `$upperString` deklariert und ihr die, in Großbuchstaben gewandelte, Zeichenkette aus `$item1` zugewiesen. Der Where-Ausdruck in Zeile 4 testet für das jeweilige `$item1`/`$item2`-Paar der Iteration, ob `$item1` das Suchwort `Parkett` enthält. Dazu wird die standardmäßige XQuery-Methode `contains` genutzt. Falls die Where-Anweisung *true* liefert, wird das `$item1`/`$item2`-Paar in die Ergebnissequenz aufgenommen. Die nachfolgende `order by`-Anweisung sortiert das Ergebnis alphabetisch nach `$item1` (Z. 5). Die Return-Anweisung schließt den FLWOR-Ausdruck ab.

Produktion 69 definiert, dass in einer Return-Anweisungen auch komplexe Unteranweisungen, wie weitere FLWORs und XML-Elementerstellungen genutzt werden können. In Zeile 6 kann daher das XML-Element `item` deklariert werden. Dessen Attribute nehmen die unveränderten Elemente der Eingangssequenzen aus. Als Elementinhalt dient der Wert der `$upperString`-Variable. Sind die Element- und Attributnamen wie hier zum Zeitpunkt der Anfragedefinition bekannt, können *direkte Konstruktoren* (engl. *direct constructors*) verwendet werden. Werden

Element- und Attributnamen hingegen aus Anfrageergebnissen dynamisch abgeleitet, müssen sogenannte *computed constructors* zur Elementdefinition eingesetzt werden (W3C, 2014d, Kap. 3.9.3). Das Ergebnis der Anfrage ist in Auflistung 4.15 dargestellt.

```
<item first="Click-Parkett" second="Variante 1">CLICK-PARKETT</item>
<item first="Click-Parkett" second="Variante 2">CLICK-PARKETT</item>
<item first="Parkett" second="Variante 1">PARKETT</item>
<item first="Parkett" second="Variante 2">PARKETT</item>
```

Auflistung 4.15: Die sich aus Anfrage 4.14 ergebenden XML-Elemente

Innerhalb eines FLWOR-Ausdrucks können Joins realisiert werden. Dazu wird aus mindestens zwei Eingangssequenzen das kartesische Produkt erzeugt. In einer nachfolgenden Where-Anweisung werden Attributwerte der Iterationselemente auf Gleichheit untersucht. Durch dieses Vorgehen kann ein Equi-Join realisiert werden. Auch die Ausführung eines äußeren Joins ist möglich. Hierfür müssen die Schlüsselwörter `allowing empty` genutzt werden.

4.5.3 Musteranfragen in XQuery

Als XQuery-Implementierung wird im Rahmen dieser Arbeit das *BaseX System* verwendet. Die Open Source-Anwendung wird von der Arbeitsgruppe *Databases and Information Systems* an der Universität Konstanz entwickelt (Grün, 2015). BaseX ist als Standalone- und als Server/Client-Anwendung einsetzbar und bietet einen XQuery 3.0 Prozessor, der für die Verarbeitung umfangreicher XML-Datenbestände optimiert wurde.

Wie bereits beschrieben, existiert neben dem *Clear Text Encoding* der ISO:10303-21 mit ifcXML zusätzlich eine XML-basierte Repräsentation für Gebäudemodelle (siehe Kap. 3.2.2). Mit der Verfügbarkeit eines standardisierten XML-Formats bietet sich XQuery als direkt nutzbare Anfragesprache für die Analyse von Gebäudemodellen an. Eine Transformation zwischen dem Datenmodell der Eingangsdaten und dem der Anfragesprache muss nicht durchgeführt werden. Als Datenquelle für die nachfolgenden XQuery-Anfragen kann somit eine ifcXML4-Datei dienen. Dazu wurde die ursprüngliche P21-Datei des translaTUM-Projekts entsprechend konvertiert (vgl. Kap. 3.3.8) .

Auflistung 4.16 zeigt Musteranfrage 1 (Wände) in XQuery. Zu Beginn werden durch einen XPath-Ausdruck alle eigenständigen Entitäten der Datenbasis an die Variable `$uos` gebunden (Z. 1). Durch eine weitere Pfadangabe werden in der folgenden Iteration ausschließlich XML-Elemente mit dem Namen IfcWallStandardCase selektiert (Z. 2). Ein Ausschnitt aus dem Anfrageergebnis ist in Auflistung 4.17 wiedergegeben.

```
1 let $uos := db:open("TranslaTUM","TranslaTUM.ifcxml")/iso_10303_28/uos
2 for $wall in $uos/IfcWallStandardCase
3 return $wall
```

Auflistung 4.16: Musteranfrage 1 (Wände) in XQuery

```
<IfcWallStandardCase id="i2145">
  <GlobalId>0C8TTVkBr1sOzkwRFKLMds</GlobalId>
  <Name>Basic Wall:Stahlbeton WU 40.0:149575</Name>
  [weitere IfcWallStandardCase Attribute]
</IfcWallStandardCase>
[2262 weitere IfcWallStandardCase Elemente]
```

Auflistung 4.17: Ausschnitte aus dem Ergebnis von Musteranfrage 1 (Wände)

Im Folgenden wird Musteranfrage 2 (Wand/Tür-Paare) in XQuery umgesetzt (Auflistung 4.18). Anstelle der `db:open`-Methode wird hier, wie in den nachfolgenden Musteranfragen, der aktuelle Kontext des XQuery-Prozessors genutzt.

```
1  for $relV in /iso_10303_28/uos/IfcRelVoidsElement,
2      $wall in /iso_10303_28/uos/IfcWallStandardCase,
3      $op in /iso_10303_28/uos/IfcOpeningElement,
4      $relF in /iso_10303_28/uos/IfcRelFillsElement,
5      $door in /iso_10303_28/uos/IfcDoor
6  where  $relV/RelatingBuildingElement/IfcWallStandardCase/@ref = $wall/@id and
7         $relV/RelatedOpeningElement/IfcOpeningElement/@ref = $op/@id and
8         $relF/RelatingOpeningElement/IfcOpeningElement/@ref = $op/@id and
9         $relF/RelatedBuildingElement/IfcDoor/@ref = $door/@id and
10        $door/OverallHeight >= 2200.
11 return <Pair>{$wall, $door}</Pair>
```

Auflistung 4.18: Musteranfrage 2 (Wand/Tür-Paare) in XQuery

Um die beschriebenen Relationen zwischen Wänden und beherbergten Bauteilen aufzulösen, werden aus der XML-Datenbasis alle Instanzen der Typen IfcRelVoidsElement, IfcWallStandardCase, IfcOpeningElement, IfcRelFillsElement und IfcDoor ausgewählt und entsprechenden Variablen zugewiesen (Z. 1-5). In den nachfolgenden vier Join-Operationen werden die Attributwerte der jeweiligen Iterationselemente untereinander verglichen und so die gesuchten Relationen aufgelöst (vergl. Abbildung 4.2). Die abschließende Return-Anweisung gibt `Pair`-Elemente zurück, in denen Wand und beherbergte Tür kombiniert werden. Um innerhalb der `Pair`-Deklaration auf Variablen der Anfrage zugreifen zu können, müssen diese in geschweiften Klammern eingebettet werden (Z. 11). Ein Ausschnitt des Ergebnisses ist in Auflistung 4.19 dargestellt.

```
<Pair>
  <IfcWallStandardCase id="i214365">
    <GlobalId>359pUVJg54ielmpVcUtSRy</GlobalId>
    <Name>Basic Wall:Leichte Trennwände 150.0:641970</Name>
    [weitere IfcWallStandardCase Attribute]
  </IfcWallStandardCase>
  <IfcDoor id="i214428">
    <GlobalId>0bg32gPqT7bhbxKM3kyZim</GlobalId>
    <Name>Single-Flush (AUS):T30 RS Tür 1260:642166</Name>
    [weitere IfcDoor Attribute]
  </IfcDoor>
</Pair>
[427 weitere Pair-Elemente]
```

Auflistung 4.19: Reduziertes Ergebnis von Musteranfrage 2 (Wand/Tür-Paare) in XQuery

Auflistung 4.20 zeigt die XQuery-Umsetzung von Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerks). Diese Anfrage teilt sich in drei sequenzielle FLWOR-Ausdrücke auf.

Der erste FLWOR-Block bildet Paare aus der kompletten Wand-Entität und dem dazugehörigen aufgabenbezogenen Element. Um die Relationen zwischen IfcWallStandardCase, IfcBuildingStorey, IfcTaskTime und den weiteren Elementen aufzulösen, bedarf es fünf Joins. Dazu werden die verwendeten Typen aus der XML-Datenbasis extrahiert (Z. 1-7). In den nachfolgenden Join-Operationen werden die entsprechenden Attributwerte verglichen und die gesuchten Relationen aufgelöst (Z. 8-12). Sowohl XML als auch XQuery unterstützen Sequenzen als Attributwerte. Dadurch muss im Gegensatz zur SQL-Variante, die ausschließlich atomare Werte verarbeiten kann, keine weitere Aufteilung der verwendeten Set-Attribute *RelatedObjects* und *RelatedElements* durchgeführt werden. Innerhalb der Where-Anweisung wird abschließend der Stockwerkname überprüft (Z. 13).

Aus den so erhaltenen Paaren aus Wand und Fertigungsende muss der Zeitwert, der als ifc:IfcDateTime Type vorliegt, mit einer Cast-Anweisung in ein xs:date umgewandelt werden (Z. 16). Dies ist nötig, um in der abschließenden Iteration die `max`-Funktion auf die Datumswerte in `$af` anwenden zu können. Mit Hilfe des Maximalwertes werden im letzten Schritt die entsprechenden Paare selektiert und durch Projektion auf Wände reduziert (Z. 18 u. 19).

```
1 let $wallTime := for $relAP in /iso_10303_28/uos/IfcRelAssignsToProduct,
2         $wall in /iso_10303_28/uos/IfcWallStandardCase,
3         $task in /iso_10303_28/uos/IfcTask,
4         $time in /iso_10303_28/uos/IfcTaskTime,
5         $st in /iso_10303_28/uos/IfcBuildingStorey,
6         $relCSS in /iso_10303_28/uos/IfcRelContainedInSpatialStructure,
7         $relAP in /iso_10303_28/uos/IfcRelAssignsToProduct
8       where $relAP/RelatedObjects/IfcTask/@ref = $task/@id and
```

```
 9           $relAP/RelatingProduct/IfcWallStandardCase/@ref = $wall/@id and
10           $task/TaskTime/IfcTaskTime/@ref = $time/@id and
11           $relCSS/RelatingStructure/IfcBuildingStorey/@ref = $st/@id and
12           $relCSS/RelatedElements/IfcWallStandardCase/@ref = $wall/@id and
13           $st/Name = "UG3"
14         return <Pair >{$wall, $time}</Pair>
15 let $af := for $t in $wallTime/IfcTaskTime/ActualFinish
16         return (xs:date($t))
17 for $wt in $wallTime
18         where $wt/IfcTaskTime/ActualFinish = max($af)
19         return $wt/IfcWallStandardCase
```

Auflistung 4.20: Musteranfrage 3 (Zuletzt fertiggestellte Wände eines Stockwerk) in XQuery

Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) lässt sich in Standard-XQuery nicht umsetzen.

Die Kombination des ifcXML-Formats und der Anfragesprache XQuery ermöglicht die grundlegende Analyse von Gebäudemodellen. Eine Transformation zwischen der IFC-Modellierung und dem von der Anfragesprache genutzten Datenmodell muss nicht erfolgen. Damit hebt sich XQuery von SQL, LINQ und der nachfolgend betrachteten Anfragesprache SPARQL ab. Da die Modelltransformationen in den letztgenannten Sprachen in unterschiedlichen Varianten durchgeführt werden können, sind entsprechende Anfragen im Allgemeinen nicht universell einsetzbar, sondern müssen von System zu System angepasst werden.

Durch die Standardisierung von ifcXML und XQuery wird hingegen die Wiederverwendbarkeit von Anfragen auf unterschiedlichen Systemen garantiert. Vorteilhaft erweist sich zudem die direkte Unterstützung von containertypisierten Attributen in FLWOR-Ausdrücken.

Auf der anderen Seite benötigt der Anwender neben Kenntnissen über die IFC-Modellierung auch fundiertes Wissen bezüglich XQuery inklusive XPath, der Extensible Markup Language und der XML Schema Definition Language. Beispiele für komplexe Anfragedefinitionen sind die benötigten Cast-Operationen und die Syntax zur Generierung neuer XML-Elemente unter Verwendung von Anfragevariablen.

4.6 Analyse auf Basis von Linked Data-Modellierung

Netzwerk- und internetbasierte Technologien erlauben den Zugang zu weitreichenden, verteilten Datenbeständen. Derartige Technologien haben sich auch im Bauwesen auf die Datenhaltung und Datenerfassung ausgewirkt (Nitithamyong & Skibniewski, 2006). Dabei ist die

Struktur, in der diese verteilten Daten vorliegen, primär für die computergestützte Präsentation und eine nachfolgende Interpretation durch menschliche Anwender ausgelegt.

Auf Grund einer fehlenden Möglichkeiten zur Vernetzung und formalen Beschreibung der Dateninhalte erschweren derartige Formate eine automatisierte Prozessierung sowie Interpretation und verhindern damit eine maschinelle Verarbeitung. Diese Problematik tritt unter anderem bei der Nutzung von HTML-Dokumenten, dem primären Format für die Repräsentation von Internetseiten, auf. Auflistung 4.21 zeigt einen Ausschnitt aus einer beispielhaften Bauteilliste innerhalb einer HTML-Instanzdatei.

```
<ul>
  <li>Basic Wall:Leichte Trennwände 150.0:212212</li>
  <li>Basic Wall:Leichte Trennwände 150.0:212287</li>
  <li>Single-Flush:T30 Tür 1010:640221</li>...
</ul>
```

Auflistung 4.21: Aufzählung von Bauteilen, Ausschnitt aus einem HTML-Dokument

Durch die genutzten HTML-Tags wird eine Aufzählung deklariert. Die sich ergebene Datenstruktur weist jedoch mehrere Defizite auf. Somit ist eine einheitliche Beziehungsmodellierung zwischen Elementen unterschiedlicher Dokumente nicht möglich. Eine semantische Interpretation der Daten und weitere Rückschlüsse können auf Basis dieser Modellierung ebenfalls nicht erfolgen.

Um die Wiederverwendung und rechnergestützten Verarbeitung von Daten zu erreichen, müssen einheitliche Regeln für die verwendeten Datenstrukturen vorliegen. Derzeit erfolgen die webbasierte Datenprozessierung über Standards wie XML und der *JavaScript Object Notation (JSON)*. Besonders XML wurde als generelle Lösung für die beschriebene Interoperabilität zwischen verteilten Datensätzen gesehen. Jedoch ergaben sich durch die Technologie lokal geltende Schemata, die untereinander nicht in Beziehung stehen. Um diese zu verknüpfen bedarf es manuelle Arbeitsschritte, wodurch eine automatisierte Verarbeitung verhindert wird (Beetz et al., 2009, S. 90).

Daher wird angestrebt von der derzeitigen dokumentenzentrischen zu einer datenzentrischen Modellierung zu wechseln. In dieser werden nicht mehr Dokumente miteinander verknüpft, sondern feingranulare Beziehungen auf Entitätsebene vorgehalten. Ein weiterer Unterschied besteht darin, dass derartige Beziehungen typisiert vorliegen. Unter Verwendung einheitlicher Standards ist dadurch die Entwicklung eines gemeinschaftlichen, globalen Informationsraums möglich, auf dem generische Applikationen arbeiten können. In diesem Kontext spricht man von *Linked Data* (Bizer et al., 2009; Heath & Bizer, 2011, S. 5).

Neben der einheitlichen Strukturierung und Vernetzung von Daten ist deren Bedeutung entscheidend. Um Linked Data-Inhalte automatisiert verarbeiten zu können, muss daher die

Semantik einzelner Datenelemente formal vorgehalten werden (Shadbolt et al., 2006). Dadurch soll die Entwicklung neuartiger, vernetzter Softwaresysteme ermöglicht werden. Die sich ergebende Struktur wird *semantic web* bezeichnet (Berners-Lee et al., 2001).

Um die Semantik von Daten formal zu beschreiben, werden *Ontologien* genutzt. Im Kontext der Informatik definiert eine Ontologie den nutzbaren Wortschatz innerhalb eines abgeschlossenen Wissensbereiches (Gruber, 1993). Zusätzlich werden Aussagen über mögliche Beziehungen zwischen Entitäten getroffen. Auf Basis dieser Modellierung ermöglichen Ontologien *computerbasiertes Folgern* (engl. *reasoning*), wodurch neue Informationen automatisiert erschlossen werden können.

Das Semantic Web benötigt ein Datenmodell, das eine globale Vernetzung von Inhalten und eine maschinenlesbaren Semantik ermöglicht. Die W3C hat mit dem *Resource Description Framework (RDF)* ein entsprechendes Modell standardisiert (Cyganiak et al., 2014). Die RDF-Spezifikation beinhaltet neben dem primären, konzeptionellen Modell auch die Beschreibung mehrere Serialisierungsformate. Dazu gehörten unter anderem *N-Triples* und *Turtle* (W3C, 2014a,b). Mit der *Web Ontology Language (OWL)* können Ontologien definiert und innerhalb des RDF-Modells zugänglich gemacht werden (Bao et al., 2012). *RDF Schema* ist eine Sprache zur Erstellung von RDF-Schemata. SPARQL Protocol and RDF Query Language (SPARQL) verwirklicht eine Anfragesprache für RDF-Inhalte (W3C, 2013, 2014c).

Eine Reihe von Forschungsarbeiten hat sich mit der Übertragung von Methoden aus dem Bereich des semantischen Webs auf das Bauwesen befasst. Von Torma (2013) wird diesbezüglich die Verknüpfung von Teilmodellen untersucht, um auf diese Weise die integrierte Nutzung unterschiedlicher Sichten eines Gebäudemodells zu ermöglichen. Die Entwicklung eines Systems zum Wissensmanagement im Bauwesen auf Basis von Ontologien wird von Lima et al. (2005) beschrieben. Ebenfalls mit Ontologien beschäftigt sich Beetz et al. (2009). Am Beispiel des IFC-Modells wird hier die Ableitung von Ontologien für EXPRESS-basierte Datenmodelle behandelt.

4.6.1 Grundlagen des RDF-Modells und SPARQL

Eine Instanz des RDF-Datenmodells, auch als *RDF-Graph* bezeichnet, besteht aus einer Menge an *Tripeln*. Jedes Tripel wird aus einem Subjekt und einem Objekt gebildet. Zwischen diesen beiden Elementen wird über ein Prädikat eine Beziehung definiert. Das Prädikat ist gerichtet und zeigt naturgemäß vom Subjekt zum Objekt. Im RDF-Graphen stellen Subjekt und Objekt Knoten dar, das Prädikat wird als Kante abgebildet. Ein komplettes Tripel wird als *RDF-Aussage* (engl. *RDF statement*) bezeichnet und dient als primäre Informationseinheit des Datenmodells (siehe Abbildung 4.7).

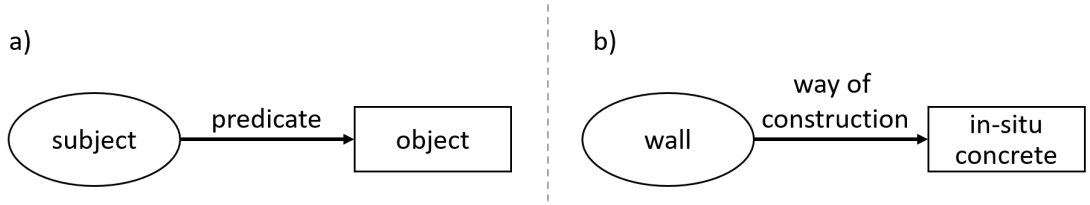


Abbildung 4.7: a) Aufbau einer RDF-Aussage, b) beispielhafte RDF-Aussage

Eine wesentliche Eigenschaft des angestrebten semantischen Webs ist die Verknüpfungen von Daten, die aus unterschiedlichen Quellen stammen. Dafür werden weltweit eindeutige Bezeichnungen benötigt, um einzelne Bestandteile von RDF-Aussagen unmissverständlich anzusprechen. Dazu kann einem RDF-Element ein eindeutiger *Internationalized Ressource Identifier (IRI)* zugeordnet werden (W3C, 2001b,a).

Die *Web Ontology Language (OWL)* ist, ihrem Namen entsprechend, eine Sprache zur Definition von Ontologien für das semantische Web. Des Weiteren stellt OWL2, die aktuelle Version der Sprache, mehrere Austauschformate mit jeweils eigener Syntax bereit. Nach dem Konzept von OWL2 kann eine Ontologie als abstrakte Struktur oder als RDF-Graph angesehen werden. Mit *direct semantics* und *RDF-based semantics* existieren zwei Möglichkeiten, um die Bedeutung von Elementen zu definieren.

Durch direct semantics werden ontologische Strukturen selbst mit Semantik versehen. Hierdurch entstehen Strukturen die bestimmter *deskriptiver Logik (DL)*, Teil der *Prädikatenlogik erster Stufe*, genügen (Horrocks et al., 2006). Durch die RDF-basierte Semantik wird die Bedeutung hingegen an die Struktur des RDF-Graphens gebunden und geht indirekt an die Elemente der Ontologie über. Die erste Variante der Bedeutungsdefinition kann nur bei bestimmten Strukturen angewendete werden. Diese werden als *OWL DL-Ontologien* bezeichnet. Die graphbasierte Bedeutungszuordnung kann hingegen bei jeder OWL2-Ontologie genutzt werden. Abbildung 4.8 illustriert die beschriebenen Hauptbestandteile der Sprache. Für eine detailliertere Erläuterung der unterschiedlichen Semantiken in OWL2 sei auf (Bao et al., 2012, Kap. 2) verwiesen.

Wie erwähnt wurde von Beetz et al. (2009) das IFC-Schemas in IfcOWL überführt. Ein verfolgtes Ziel dieser Arbeit bestand darin, nur bestimmte Konstrukte der Prädikatenlogik erster Stufe zu nutzen. Somit kann ein Kompromiss in der Ausdrucksstärke und der effizienten Verarbeitung der Ontologie erreicht werden. Außerdem können vorhandene Reasoning-Algorithmen eingesetzt werden.

Im Folgenden wird auf die Überführung von Elementen der EXPRESS-Modellierung näher eingegangen. Für jede Entitätsdefinition des Schemas wird in der Ontologie eine owl:Class eingeführt. Die Modellierung der ursprünglichen Vererbungshierarchie geschieht über

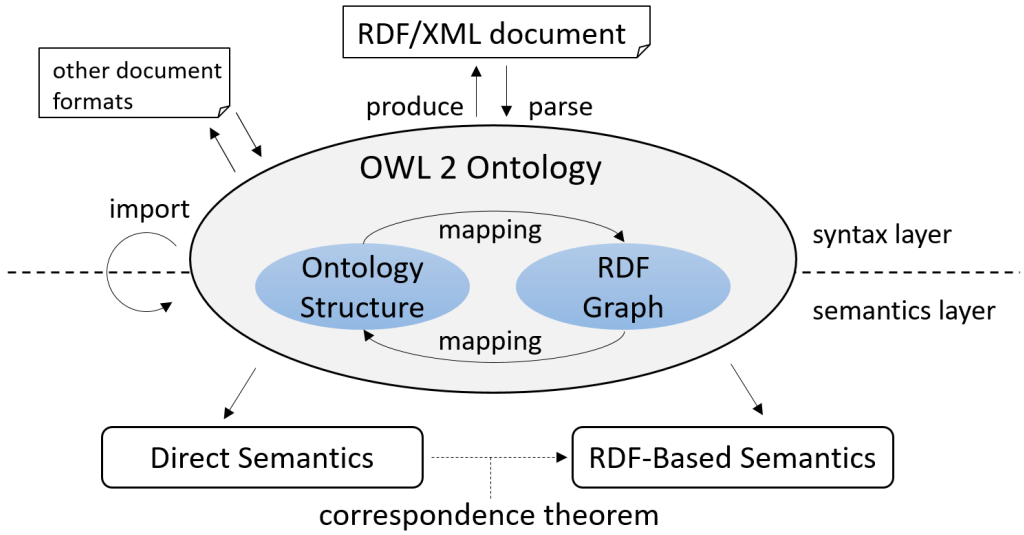


Abbildung 4.8: Die Struktur der Web Ontology Language, nach (Bao et al., 2012, Kap. 2)

`rdfs:subClassOf`-Prädikate. Auflistung 4.22 zeigt eine beispielhafte Modellierung der ontologischen Entsprechung für IfcDoor.

```
:IfcDoor
  a owl:Class;
  rdfs:subClassOf :IfcBuildingElement;
  owl:disjointWith :IfcBeam, :IfcColumn, [...];
```

Auflistung 4.22: ifcOWL-Repräsentation von IfcDoor

Die in Kapitel 3.2.1 beschriebenen einfachen EXPRESS-Typen wie REAL und STRING lassen sich vorteilhaft mit Wrapperklassen abbilden, die den zugrundeliegenden Datentyp kapseln. Eine direkte Zuordnung einfacher EXPRESS-Datentypen zu OWL-Typen ist hingegen wegen Restriktionen in Mengenvariablen problematisch (Beetz et al., 2009, Kap. 3.3).

Bei der Überführung von Attributen in eine OWL-Repräsentation sind deren Geltungsbereiche zu beachten. In EXPRESS besitzen Entitätsattribute einen lokalen Geltungsbereich, in OWL hingegen gilt ein globaler Bereich. Gleichnamige Attribute müssen daher in ein owl:ObjectProperty überführt werden, das mehrere Typen aufnehmen und unterschiedliche Datenbereiche abdecken kann. Eine owl:Class mit einem derartigen Attribut spezifiziert die zum IFC-Schema konforme Nutzung des owl:ObjectPropertys. Namenskonflikte können außerdem durch die Aufteilung des IFC-Schemas in unterschiedliche Namensräume vermieden werden.

Enumerationen des IFC-Schemas können durch *NamedIndividual*-Elemente ausgedrückt werden. Jedes Element nimmt dafür einen Wert der Enumeration auf. OWL DL-Ontologien bieten nur eingeschränkte Möglichkeiten, um geordnete Mengen zu repräsentieren. Dies erschwert

die Abbildung von Container-Typen des EXPRESS-Schemas. Eine Möglichkeit zur Repräsentation von geordneten Containern ist die Verwendung einer owl:Class mit entsprechenden Attributen. Nach dem Prinzip einer *Linked List* beinhaltet das erste Attribut den aktuellen Wert im Container und das zweite Attribut zeigt zum nächsten Teil der Sequenz.

Die unterschiedlichen Elemente der IFC- und ifcOWL-Modellierung sind in Tabelle 4.4 dargestellt. Für eine ausführliche Behandlung von IfcOWL sei zusätzlich auf (Pauwels & Terkaj, 2016) verwiesen. Im Gegensatz zu (Beetz et al., 2009) wird hier das aktuelle OWL in Version 2 behandelt. In (Pauwels & Roxin, 2016) wird zudem darauf eingegangen wie IfcOWL für bestimmte Einsatzfälle vereinfacht werden kann.

	IFC	Linked Data
Spezifikationstyp	Schema	Ontologie
Einfache Typen	TYPE	owl:Class und owl:DatatypeProperty restriction
Eingegrenzte Typen	User-defined Types	owl:class
Container Typen	SET, LIST, ARRAY	owl:class
Selektionstypen	SELECT Types	rdfs:subClassOf für owl:classes
Enumerationen	ENUMERATION	rdf:type für owl:NamedIndividuals

Tabelle 4.4: Vergleich der Modellierungselemente in EXPRESS und OWL, nach (Pauwels & Terkaj, 2016), vereinfacht

Um IFC-Instanzdateien in einen RDF-Graph zu überführen sind mehrere Anwendungen verfügbar. Im Rahmen dieser Arbeit wird das *IFC-to-RDF conversion tool 1.0* verwendet (Törmä et al., 2014). Eine weitere Implementierung ist mit *EXPRESStoOWL* verfügbar (Terkaj & Šojić, 2015).

In Auflistung 4.23 ist ein Ausschnitt der sich ergebenden OWL-Struktur im N-Triples-Format dargestellt. Dieser beinhaltet eine IfcWallStandardCase-Entität mit den Attributen *objectPlacement* und *name*. Als Eingangsquelle diente das in Kapitel 3.3.8 beschriebene TranslaTUM-Modell, das mit Hilfe der oben genannten Anwendung konvertiert wurde. Ab der zweiten Aussage sind aus Darstellungsgründen Präfixe verwendet worden.

```
1 <http://linkedbuildingdata.net/model/GUID_KhP5RMaBRjqKuJp_yQDofQ>
2   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
3     <http://linkedbuildingdata.net/schema/IFC4#IfcWallStandardCase> .
4 lbd:GUID_KhP5RMaBRjqKuJp_yQDofQ lbd:objectPlacement _:AX5fXLINEX5fX335978 .
5 lbd:GUID_KhP5RMaBRjqKuJp_yQDofQ lbd:name "Fassade"^^xs:string .
```

Auflistung 4.23: Ausschnitt aus der N-Triples-Daten des konvertierten TranslaTUM-Modells

Der Ausschnitt beinhaltet sogenannte *blank nodes*, die mit _: eingeleitet werden. Diese Bezeichner sind nur im lokalen Geltungsbereich eines Datenbestandes eindeutig (vgl.

objectPlacement-Attribut, Z. 4). Durch die Zeichen ^^ kann der Datentyp eines Knotens vorgehalten werden. Hierfür können unter anderem die Typen der XML Schema Definition Language genutzt werden. So wird das *name*-Attribute in Zeile 5 als xs:string-Typ ausgezeichnet.

Durch diese umfangreiche explizite Auszeichnung, in der jeder Attributwert durch ein RDF-Statement vorgehalten wird, steigt die auszuwertende Datenmenge. Im konkreten Beispiel beträgt die Dateigröße des ursprünglichen IFC-Modells 22,8 MB. Die generierte N-Triples-Datei weist ein Größe von 349,5 MB auf. Dies entspricht einem Größenverhältnis von $\sim 1:15$.

Anstelle einer dateibasierten Speicherung bieten sich für ausgedehnte Datenbestände die Nutzung RDF-optimierter Datenbanken an. Eine derartige Datenbank wird auch als *TripleStore* bezeichnet. Die Implementierung der Musteranfragen erfolgte in der *Brightstar*-Datenbank mit dem *Polaris*-DBMS. Letzteres stellt den benötigten Interpreter für die Anfrage- und Manipulationssprache SPARQL bereit (Ahmed & Moore, 2014).

Die Filterung einer RDF-Datenbasis kann in SPARQL durch eine Kombination aus `SELECT`- und `WHERE`-Anweisungen geschehen. Innerhalb des `WHERE`-Teils werden Prädikate durch sogenannte *Triple-Pattern* ausgedrückt. Diese Muster können Variable und Konstante beinhalten, wobei Variable mit einem `?`-Zeichen beginnen müssen. Für eine umfassende Beschreibung der SPARQL-Syntax wird auf die entsprechenden W3C-Spezifikation verwiesen (W3C, 2008, Kap. 4).

4.6.2 Musteranfragen in SPARQL

Auflistung 4.24 zeigt die Umsetzung von Musteranfrage 1 (Wände) in SPARQL. Im RDF-Datenmodell existiert keine Struktur, um aggregierende Entitäten abzubilden. Eine komplexe IFC-Entität, wie IfcWallStandardCase besteht aus einer ungruppierten Menge an RDF-Statements. Um die entsprechenden Statements auszuwählen, wird in Zeile 2 ein Triple-Pattern mit drei Variablen definiert. Anschließend werden die Tripel ausgewählt, in denen das Subjekte `?wall` enthalten und als lbd:IfcWallStandardCase typisiert ist. Durch die Nutzung des `;`-Zeichens im entsprechenden `WHERE`-Teil muss das gleichbleibende Subjekt nicht wiederholt werden (Z. 3). Das Ergebnis der Anfrage ist in Auflistung 4.25 dargestellt. Auf die Auszeichnung des `lbd`-Präfix wird in den kommenden Anfragen verzichtet.

```
1 PREFIX lbd: <http://linkedbuildingdata.net/schema/IFC4#>
2 SELECT ?wall ?predicate ?wallprop
3 WHERE { ?wall rdf:type lbd:IfcWallStandardCase; ?predicate ?wallprop . }
```

Auflistung 4.24: Musteranfrage 1 (Wände) in SPARQL

```
1 [?wall]                    [?predicate]    [?wallprop]
2 GUID_B8MUE2csRFSLeyGIau2ICQ type            IfcWallStandardCase
3 GUID_B8MUE2csRFSLeyGIau2ICQ name            Basic Wall:Daemmstoff 50.0:846013
4 GUID_B8MUE2csRFSLeyGIau2ICQ tag             846013
5 [5 weitere Tripel für GUID_B8MUE2csRFSLeyGIau2ICQ, 2260 weitere Entitäten]
```

Auflistung 4.25: Teilergebnis von Musteranfrage 1 (Wände) in SPARQL

Um Musteranfrage 2 (Wand/Tür-Paare) zu realisieren, werden durch fünf Triple-Pattern die benötigen Elemente entsprechend ihre IFC-Typisierung ausgewählt (Auflistung 4.26, Z. 3-5). In den Zeilen 6 und 7 werden die über Attributwerte vorgehaltenen Relationen zwischen Wand, Öffnung und Tür geprüft. Um Operatoren der Prädikatenlogik anwenden zu können, muss in SPARQL eine `FILTER`-Anweisung innerhalb des `WHERE`-Blocks genutzt werden. Dazu wird die Türhöhe in der Variable `?he` zwischengespeichert, die anschließend innerhalb des Filters ausgewertet werden kann (Z. 8).

```
1 SELECT ?wall ?door
2 WHERE {
3   ?relV a lbd:IfcRelVoidsElement . ?wall a  lbd:IfcWallStandardCase .
4   ?op   a lbd:IfcOpeningElement .  ?relF a  lbd:IfcRelFillsElement .
5   ?door a lbd:IfcDoor .
6   ?relV lbd:relatingBuildingElement ?wall; lbd:relatedOpeningElement ?op .
7   ?relF lbd:relatingOpeningElement  ?op;   lbd:relatedBuildingElement ?door .
8   ?door lbd:overallHeight           ?he .  FILTER(?he >= 2000) .}
```

Auflistung 4.26: Musteranfrage 2 (Wand/Tür-Paare) in SPARQL

Wie in den bereits behandelten Anfragekonzepten, gestaltet sich die Umsetzung von Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) auch in SPARQL als aufwendig (siehe Auflistung 4.27). In dieser Anfrage müssen die Beziehungen zwischen Wänden, Stockwerken und Realisierungszeiträumen ausgewertet werden. Die sich ergebende Gesamtrelation muss daraufhin mehrfach weiterverarbeitet werden. Zum einen, um den Maximalwert des *actualFinish*-Attributs zu bestimmen und um alle Wand-Instanzen mit diesem Wert auszuwählen. Zur Vermeidung von Redundanzen in der Anfragedefinition, wird die Anfrage in zwei Teile aufgespalten.

In der ersten Teilanfrage wird eine neue Graph-Struktur `wallTime` in den Triple Store eingefügt. Diese besteht aus einer Zuordnung von Wänden innerhalb des ersten Stockwerks und dem Zeitpunkt ihrer Fertigstellung (Z. 3). Die Instanzen der relevanten IFC-Klassen werden in den Zeilen 5 bis 8 durch entsprechende Triple-Pattern in der Datenbasis identifiziert. Danach werden die entsprechenden Relationen untersucht (Z. 9 - 11). Zur Prüfung des Stockwerknamens auf die Konstante `FirstFloor` muss außerdem der in der Datenbasis verwendete

XSD-Typ, hier xsd:string, angegeben werden (Z. 11). Der Zeitpunkt der Fertigstellung wird in Zeile 12 der Variable `?acf` zugewiesen.

In der zweiten Teilanfrage wird die erstellte Graph-Struktur `wallTime` durch eine `FROM`-Anweisung als Datenquelle ausgewählt (Z. 15). Anschließend werden die erhaltenen Wand/Zeitpunkt-Statements auf diejenigen reduziert, die den maximalen *actualFinish*-Wert aufweisen. Dazu wird eine `FILTER`-Anweisung genutzt (Z. 18). Um den Maximalwert der Zeitangabe zu berechnen, folgt in Zeile 19 ein verschachtelter `SELECT-WHERE`-Block, der die Aggregationsfunktion `max` nutzt.

```
1  PREFIX cms: <http://cms.bgu.tum.de/queries#>
2  INSERT {
3    GRAPH cms:wallTime {?wal lbd:actualFinish ?acf}
4    WHERE {
5      ?relCSS a lbd:IfcRelContainedInSpatialStructure .
6      ?relAP a lbd:IfcRelAssignsToProduct .
7      ?st a lbd:IfcBuildingStorey . ?wall a lbd:IfcWallStandardCase .
8      ?task a lbd:IfcTask . ?time a lbd:IfcTaskTime .
9      ?relCSS lbd:relatedElements ?wall; lbd:relatingStructure ?st .
10     ?relAP lbd:relatingProduct ?wall; lbd:relatedObjects ?task .
11     ?st lbd:name "FirstFloor"^^xsd:string . ?task lbd:taskTime ?time .
12     ?time lbd:actualFinish ?acf .};
13 SELECT ?wal
14 FROM cms:wallTime
15 WHERE {
16   ?wal lbd:actualFinish ?acf .
17   FILTER (?acf = ?acm)
18   SELECT max(?acf) AS ?acm WHERE { ?wal lbd:actualFinish ?acf . }}
```

Auflistung 4.27: Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) in SPARQL

Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) lässt sich in Standard-SPARQL nicht umsetzen.

SPARQL ermöglicht durch das Konzept der Triple-Pattern einen neuartigen Ansatz zur Filterung komplexer Gebäudemodelle. In einem Muster können sowohl Beziehungen zwischen Entitäten als auch Attribute ausgewertet werden. Ersteres erweist sich bei der Analyse von relationsintensiven IFC-Gebäudemodellen als vorteilhaft. Auch containertypisierte Attribute werden direkt unterstützt. Durch die feingranulare Modellierung und die sich daraus ergebenden vielfachen Variablendeklarationen erfordern jedoch bereits einfache Analysen umfangreiche Anfragen. Für Nutzer ohne Programmierkenntnisse wird das Erstellen von SPARQL-Anfragen außerdem durch die teilweise benötigten Typangaben erschwert. Zudem verhindert das Konzept des RDF-

Datenmodells ohne eine übergeordnete, aggregierende Objektstruktur die Extraktion komplexer Entitäten als eigenständige Einheiten. Abschließend sei nochmals auf den hohen Speicherbedarf für RDF-basierte Gebäudemodelle hingewiesen, der um den Faktor 15 höher sein kann als in der originären IFC-Modellierung.

4.7 Erweiterungen für räumliche und zeitliche Analysen

Die bisher umsetzbaren Musteranfragen beinhalten keine räumlichen Analysen, die auf einer ausschließlich geometrischen Auswertung beruhen. Das Enthaltensein von Bauteilen in räumlichen Strukturen des Gebäudes (Musteranfrage 3) erfolgte durch die Verarbeitung des objektivierten Relationstyps IfcRelContainedInSpatialStructure. Dahingegen sollte es möglich sein implizit vorgehaltene, räumliche Sachverhalte durch Erweiterungen der eingesetzten Analysesysteme und -sprachen erschließen zu können.

Für zweidimensionale Geometrierepräsentationen ist eine derartige geometrische Funktionalität bereits standardisiert worden. Auch für die Verarbeitung zeitlicher Daten existieren standardisierte Ansätze. Im Folgenden wird auf derartige raum-zeitliche Erweiterungen für SQL und SPARQL eingegangen.

Unter Verwendung der *ISO/IEC-13249* definiert die bereits erwähnte Simple Feature Access Specification SQL-Geometrietypen und -operationen (ISO, 2016a). Die enthaltenen topologischen Operationen beruhen auf dem DE9-IM (ISO, 2010). Derzeitige Implementierungen realisieren meistens eine Konformitätklasse für zweidimensionale Feature, wohingegen 3D-Operationen nur teilweise unterstützt werden (PostGIS, 2016). Ein Beispiel einer Simple Feature-konformen SQL-Anfrage ist in Auflistung 4.28 dargestellt. Diese wurde (ISO, 2010, S. 110) entnommen und testet, ob sich die Mittelachsen zweier Straßensegmente schneiden.

```
1 SELECT Intersects(road_segments.centerline, divided_routes.centerlines)
2 FROM road_segments, divided_routes
3 WHERE road_segments.fid = 102 AND divided_routes.name = 'Route 75';
```

Auflistung 4.28: Räumliche Analyse in SQL auf Basis der ISO-19125

Um räumliche Analysen in RDF-Daten zu vereinheitlichen, existiert für SPARQL mit *GeoSPARQL* ein vergleichbarer Standard (Perry & Herring, 2012). Neben den topologischen Operationen der ISO-19125 ist in der GeoSPARQL-Spezifikation eine Ontologie für Feature- und Geometrietypen enthalten. Außerdem werden Transformationsregeln für räumliche Anfragen definiert, um diese auf Basis räumlicher Indizierung performanter ausführen zu können.

Auflistung 4.29 zeigt eine GeoSPARQL-Anfrage, die aus (Battle & Kolas, 2012, Seite 12) entnommen wurde. Die Anfrage identifiziert alle Geometrieelemente innerhalb von zehn Kilometern um Feature `4212826`. Dabei werden die Funktionen `geof:buffer` und `geo:sfContains` genutzt. Die erste Funktion erstellt eine Puffergeometrie um das genannte Feature. Die zweite Funktion testet welche RDF-Subjekte innerhalb des Puffers liegen. Das Schlüsselwort `BIND` dient hierbei zur Verknüpfung der Variable `?buff` mit der erstellten Puffergeometrie.

```
SELECT ?x
WHERE { GRAPH <http://www.geonames.org> {
  <http://sws.geonames.org/4212826/> geo:hasGeometry ?geo1 .
  ?geo1 geo:asWKT ?wkt1 .
  BIND (geof:buffer(?wkt1, 10000, units:m) as ?buff) .
  ?x geo:hasGeometry ?geo2 .
  [ a geo:Geometry ; geo:asWKT ?buff ] geo:sfContains ?geo2 .}}
```

Auflistung 4.29: Räumliche Analyse in GeoSPARQL auf Basis der ISO-19125

Temporale Datenbanken sind primär für die Vorhaltung und Verarbeitung sich zeitlich verändernder Daten bestimmt (siehe Kap. 2.2.4). Mit SQL:2011 wurden entsprechende Erweiterungen in den SQL-Standard übernommen (Kulkarni & Michels, 2012). Im BIM-Bereich ist hingegen die Analyse von statischen zeitlichen Daten der baulichen Terminplanung von Interesse. Entsprechende Zeitpunkte und -Intervalle können über Vergleichsoperationen untersucht werden, wenn geeignete Datentypen und Operatoren verfügbar sind. Dies wurde mit Musteranfrage 3 (zuletzt fertiggestellte Wände in Stockwerk) bereits demonstriert.

Mit SQL:2011 stehen zusätzlich Operatoren auf einem höheren Abstraktionsniveau zur Verfügung. Diese analysieren topologische Relationen zwischen Zeitpunkten und -Intervallen. Das Konzept ist mit den Operatoren aus (Vilain, 1982) vergleichbar, auch wenn sich die Semantik einzelner Operatoren unterscheidet. In Auflistung 4.30 ist eine Anfrage auf Basis von Vergleichsoperatoren (Z. 1 u. 2) und eine weitere Anfrage mit topologischer `OVERLAPS`-Prüfung (Z. 4 u. 5) dargestellt.

```
1 SELECT * FROM IfcTaskTime
2 WHERE ActualStart < DATE '2011-01-01' AND ActualFinish > DATE '2010-01-01'
3
4 SELECT * FROM IfcTaskTime
5 WHERE ActualPeriod OVERLAPS PERIOD (DATE '2010-01-01', DATE '2011-01-01')
```

Auflistung 4.30: Zeitliche Analyse in SQL:2011

Neben diesen standardisierten Ansätzen existieren vielfältige Erweiterungen für die bereits besprochenen Anfragesprachen. Derartige Erweiterungen werden über Bibliotheken in die jeweiligen Systeme importiert und stehen in Anfragen als Funktionen bzw. in objektorientierten Systemen als Methoden bereit. Darauf wurde für den Fall räumlicher topologischer Analysen

bereits im Kontext benutzerdefinierte Datentypen in SQL eingegangen (vgl. Kap. 4.4.3). Ein Ansatz einer zeitlichen Erweiterung durch Intervallprädikate für SPARQL ist in (Bereta et al., 2013) beschrieben. In LINQ bietet sich die Nutzung eines benutzerdefinierten Intervalltyps unter Verwendung von .NET-Klassen wie DateTime und TimeSpan an. Auch XQuery lässt sich über entsprechende Funktionen erweitern (Walmsley, 2007, Kap. 8).

Die standardisierten räumlichen Erweiterungen in SQL und SPARQL eignen sich nur bedingt für eine spatiale Analyse von Gebäudemodellen, da diese primär auf zweidimensonale Repräsentationen ausgelegt sind. Ein Ansatz für die Verarbeitung datenintensiver dreidimensionale Modelle fehlt hingegen. Zeitliche Analysen lassen sich einfacher umsetzen, da diese durch Vergleichsoperatoren umsetzbar sind. Hierbei ergeben sich jedoch mitunter umfangreiche Anfragen, deren Definition fehleranfällig ist. Durch höherwertige topologische Operatoren auf Zeitintervallebene kann die Definitionen zeitlicher Analysen vereinfacht werden (vgl. Kap. 5.1).

4.8 Domänenspezifische Analysekonzepte

Neben den vorgestellten allgemeinen Anfragekonzepten befasst sich die Forschung mit domänenspezifischen Ansätzen zur Analyse und Filterung von Gebäudemodellen. Derartige Bemühungen sollen die Defizite allgemeiner Anfragesprachen bei der Auswertung von BIM-Daten ausgleichen. Im Folgenden werden relevante Entwicklungen vorgestellt. Auf die BERA-Sprachen und das BIMcraft-Projekt kann auf Grund der Art bzw. wegen des begrenzten Umfangs der verfügbaren Literatur nur beschränkt eingegangen werden.

4.8.1 PMQL

Die *Partial Model Query Language (PMQL)* ist eine Anfrage- und Datenmanipulationssprache zur Teilmodellextraktion aus serverseitig vorliegenden Gebäudemodellen (Adachi, 2002b). Eine Implementierung von PMQL existiert als Komponente des *IFC Model Servers* (Adachi, 2002c). Die Entwicklung des Servers und seiner Anfragemethodik basiert auf der These, dass eine dateibasierte Vorhaltung von Gebäudemodellen in der Praxis zu Problemen führen kann. Stattdessen wird eine Client-Server-Umgebung vorgeschlagen, in der kein dateibasierter Datenaustausch erfolgt.

Die Clients erhalten die jeweils benötigte Teilmenge aus dem am Server vorgehaltenen Gesamtmodell über *Web Services*. Dabei wird zur Übertragung von PMQL-Anfragen das *Simple Object Access Protocol (SOAP)* genutzt. In diesem Protokoll lassen sich PMQL-Anweisungen einbetten, um `Select-`, `Update-` und `Delete-`Operationen auszuführen. Da Entitäten innerhalb

der Server-Anwendung in einer relationalen Datenbank vorgehalten werden, müssen PMQL-Anweisungen zur Laufzeit in SQL-Befehle übersetzt werden.

Im Folgenden wird auf die PMQL-Grammatik eingegangen. Da PMQL als XML-Dialekt umgesetzt wurde, gelten die XML-Syntaxregeln. So muss jede Anfrage innerhalb eines einzigen Wurzelelements eingebettet werden. Dabei handelt es sich um das `<pmql>`-Element. Innerhalb diesem wird ein `<select>` eingebettet. Durch Spezifizierung des Elementattributs *action* kann eine Selektion, eine Löschung oder ein Update der gefundenen IFC-Entitäten geschehen. Außerdem kann eine Traversierung von IFC-Objektattributen erfolgen. Dazu muss das `<select>`-Element in ein `<cascades>` verschachtelt werden. Zur Traversierung können standardmäßige als auch inverse Attribute genutzt werden. Eine Einschränkung der Ergebnismenge erfolgt über `Where`-Anweisungen. Diese ermöglichen eine Filterung durch Objektbezeichner, IFC-Typen und eingebettete SQL-Anweisungen.

Die Nutzung von SQL innerhalb von PMQL demonstriert die aus (Adachi, 2002b, Kapitel 5.2) entnommene Anfrage in Auflistung 4.31. Hier werden Wände selektiert, deren *Label*-Attribut mit der Zeichenkette `Wall` beginnt.

```
1 <pmql>
2   <select type="entity" match="IfcWall" action="get">
3     <where><expr value="Label LIKE 'Wall#%'"/></where>
4   </select>
5 </pmql>
```

Auflistung 4.31: PMQL-Anfrage zur Selektion von Wänden, nach (Adachi, 2002b, Kapitel 5.2), verkürzt wiedergegeben

Das IFC Model Servers Projekt wurde im September 2002 abgeschlossen (Adachi, 2002a). Seitdem wurde PMQL nicht weiterentwickelt. Zudem ist die Implementierung des Servers nicht öffentlich zugänglich und somit keine Testumgebung für PMQL-Anfragen verfügbar. Die folgenden Musteranfragen wurden daher ausschließlich auf Basis von (Adachi, 2002b) erstellt. Auflistung 4.32 zeigt Musteranfragen 1 (Wände) in PMQL.

```
1 <pmql><select type="entity" match="IfcWall" action="get" /></pmql>
```

Auflistung 4.32: Musteranfrage 1 (Wände) in PMQL

Eine Teilumsetzung von Musteranfrage 3 (Wand/Tür-Paare) in PMQL ist in Auflistung 4.33 dargestellt. Durch verschachtelte `cascade`/`select`-Operationen kann von IfcWall über IfcRelVoidsElements, IfcOpeningElement und IfcRelFillsElement zur IfcDoor navigiert werden. Dabei wird auch die Auswertung inverser Attribute wie *HasOpenings* und *HasFillings* unterstützt. Innerhalb der letzten `select`-Anweisung wird auf den Typ der referenzierten Entität, hier auf IfcDoor, getestet (Z. 10). Die Einschränkung auf eine Mindesthöhe erfolgt in der `Where`-Anweisung in Zeile 12. Die Anfrage liefert jedoch nicht ausschließlich Wände und beherbergte

Türen, sondern alle fünf beteiligten Entitätstypen und ist somit keine strikte Umsetzung von Musteranfrage 3.

```
<pmql>
  <select type="entity" match="IfcWall" action="get">
    <cascades>
      <select type="inverse" match="HasOpenings" action="get">
        <cascades>
          <select type="attribute" match="RelatedOpeningElement" action="get
          ">
            <cascades>
              <select type="inverse" match="HasFillings" action="get">
                <cascades>
                  <select type="entity" match="IfcDoor" action="get">
                    <where>
                      <expr value="OverallHeight >= 2000"/>
                    </where>
                  </select>
                </cascades>
              <select/>
            </cascades>
          </select>
        </cascades>
      </select>
    </cascades>
  </select>
</pmql>
```

Auflistung 4.33: Teilumsetzung von Musteranfrage 3 (Wand/Tür-Paare) in PMQL

Durch das Konzept von PMQL ist die Sprache für die Analyse von Gebäudemodellen nur bedingt geeignet. So ermöglicht PMQL die Erstellung eines Teilmodells aus einem Gesamtmodell, kann aber die Struktur der zurückgelieferten Daten nicht verändern. Für eine Analyse von Gebäudemodellen bedarf es hingegen Funktionalität, um Entitäten und deren Relationen zu untersuchen und aufzubereiten. Beispielsweise ist Musteranfrage 3 (Wand/Tür-Paare) nicht umsetzbar, da Projektionen der Ergebnismenge nicht unterstützt werden. Ebenfalls existieren keine Operatoren für die Erstellung des kartesischen Produkts aus Entitätsmengen und die Ausführung von Joins. Die Definition von Anfragen gestaltet sich für Endanwender auf Grund der hier gewählten XML-Syntax als umständlich. So scheint das cascades-Element unnötig, da Anfragen auch ohne dieses eindeutig bleiben. Hinzukommt, dass in where-Konstrukten SQL-Anweisungen inkludiert werden müssen. Daher muss sich der Anwender sowohl mit der XML- als auch mit der SQL-Syntax auseinandersetzen.

4.8.2 BIMQL

Die *Building Information Model Query Language (BIMQL)* wurde im Kontext des *BIM Server*-Projekts entwickelt (Beetz et al., 2010). Die Architektur dieses Modellservers basiert auf einem Meta-Modellierungsansatz und der Verwendung einer *Key-Value*-Datenbank. Dabei wird das *Eclipse Modelling Framework (EMF)* genutzt, um für referenzierte IFC-EXPRESS-Schemata eine *Meta Object Facility (MOF)* zu generieren und ein EMF-Modell zu erzeugen. Dieses enthält für alle EXPRESS-Elemente entsprechende EMF-Repräsentationen. Gegenüber dem originären EXPRESS-Schema bietet das EMF-Modell umfangreichere Unterstützung zur automatischen Codegenerierung. Die erstellten IFC-Klassen sind Teil einer Java-Bibliothek und können als instanziierbare IFC-Repräsentationen vom BIMQL-Interpreter ausgewertet werden.

BIMQL unterstützt derzeit die Selektion von Untermengen aus einem IFC-Gesamtmodell (Mazairac & Beetz, 2013). Außerdem können Änderungen an bestehenden Entitäten durchgeführt werden. Eine Erweiterung der Sprache, inklusive Änderungs-, Erstellungs- und Löschoperationen, ist für zukünftige Versionen geplant. Einen Ausschnitt aus der BIMQL-Syntax ist in Abbildung 4.9 dargestellt.

```
BIMQL          ::= select
select         ::= 'Select' VARIABLE where? cascade* set?
cascade        ::= 'Select' VARIABLE ':=' VARIABLE ('.Attribute.' STRING | '.Property.'
                   STRING) where?
where          ::= 'Where' statement
set            ::= 'Set' VARIABLE '.Attribute.' STRING ':=' (INTEGER | REAL | STRING)
statement      ::= relation ('And' relation | 'Or' relation)*
relation       ::= relationleft ('=' relationright | '/=' relationright | '<' relationright
                   |'<=' relationright | '>='    relationright | '>' relationright)
relationleft   ::= (VARIABLE '.EntityType' | VARIABLE '.Attribute.' STRING | VARIABLE
                   '.Property.' STRING | VARIABLE PLUGIN)
relationright  ::= (INTEGER | REAL | STRING)
```

Abbildung 4.9: Teil der BIMQL-Syntax in Backus-Naur Form, aus (Mazairac & Beetz, 2013)

Das Wurzelelement `BIMQL` dient zur Auswahl einer der Hauptoperationen wie `create`, `update` und `select`. Derzeit wird ausschließlich `select` unterstützt, über das listenbasierte Rückgabevariablen deklariert werden können. Die `cascade`-Anweisung dient zur Verkettung mehrerer `select`-Teile. Über die `set`-Operation kann ein einfaches Attribut einer Entität geändert werden. Somit wird eine beschränkte Updatefunktionalität realisiert. Eine `relation`-Anweisung, die sich aus den syntaktischen Teilen `relationleft` und `relationright` zusammensetzt, definiert ein Prädikat innerhalb eines `where`-Ausdrucks. Dadurch kann beispielsweise eine Typprüfung und die Auswertung von Attributen erfolgen.

Auflistung 4.34 zeigt Musteranfrage 1 (Wände) in BIMQL. Wie in SPARQL müssen hier Variablen mit einem `$` eingeleitet werden. Nach Verarbeitung der Anfrage durch den Server steht

das Ergebnis in serialisierter Form als Download zur Verfügung. Dabei können unterschiedliche Formate, beispielsweise IFC, genutzt werden.

```
1 Select $wall
2 Where $wall.EntityType = IfcWallStandardCase
```

Auflistung 4.34: Musteranfrage 1 (Wände) in BIMQL

Auflistung 4.35 zeigt eine Teilumsetzung von Musteranfrage 3 (Wand/Tür-Paare) in BIMQL. Wie in PMQL umfasst die Ergebnismenge dabei alle traversierten Instanzen und nicht ausschließlich Wand/Tür-Paare. In Zeile 2 wird eine Typfilterung auf IfcWallStandardCase-Instanzen durchgeführt. Durch die folgenden vier `Select`-Anweisungen wird eine sukzessive Traversierung über die entsprechenden Entitätseigenschaften realisiert (Z. 3-6). Als letztes wird eine Typ- und Attribut-Prüfung durchgeführt, um die gefunden Entitäten auf Türen mit der geforderten Mindesthöhe einzuschränken (Z. 7 u. 8).

```
1 Select $wall
2 Where $wall.EntityType = IfcWallStandardCase
3 Select $relvoid := $wall.Attribute.HasOpenings
4 Select $opening := $relvoid.Attribute.RelatedOpeningElement
5 Select $relfill := $opening.Attribute.HasFillings
6 Select $element := $relfill.Attribute.RelatedBuildingElement
7 Where $element.EntityType = IfcDoor AND
8       $element.Attribute.OverallHeight >= 2000
```

Auflistung 4.35: Teilumsetzung von Musteranfrage 3 (Wand/Tür-Paare) in BIMQL

Der BIM Server bietet mehrere Möglichkeiten zur Serialisierung von BIMQL-Selektionen. Wird IFC als Ausgabeformat gewählt, erhält man bei Musteranfrage 3 eine umfangreiche Instanzdatei, da alle Entitäten mit ihrer Geometrierepräsentation in die Ergebnismenge übernommen werden. Dennoch weist die erhaltene Datei Defizite auf, die ihre Verwendung behindern. Unter anderem fehlen räumliche Strukturen wie IfcBuilding und IfcBuildingStorey, da diese nicht übertragen bzw. neu erstellt werden. Hier wird ersichtlich, dass die IFC kein spezielles Konzept für eine Teilmodell-Repräsentation bereitstellt. So muss die Struktur von Entitätsuntermengen zur validen Vorhaltung als IFC-Datei aufbereitet werden, was in aktuellen BIMQL-Implementierung jedoch nicht umgesetzt wurde.

Als domänenspezifische Anpassung beinhaltet BIMQL das Konzept der *Anfrageabkürzungen* (engl. *query shortcuts*), das es erlaubt, komplexe Anfragen zu vereinfachen. Das Konzept findet Anwendung bei der Selektion von Entitätseigenschaften, die über Merkmalslisten mit Entitäten verknüpft wurden. Nach der Schemadefinition erfordert die Auflösung einer kompletten Eigenschaftsverknüpfung vier Traversierungsschritte. Durch den query shortcut kann die Komplexität der Eigenschaftsevaluierung vom Endnutzer verborgen werden und die An-

frage in nur einer Anweisung umbesetzt werden. Auflistung 4.36 zeigt eine derartig verkürzte Anweisung, die die *volume*-Eigenschaft einer Entität selektiert.

```
Select $Var1
Where $Var1.Attribute.GlobalId = "'359pUVJg54ielmpVcUtSRy"'
Select ?Var2 := $Var1.Property.Volume
```

Auflistung 4.36: Verwendung eines Property-Shortcuts in einer BIMQL-Anfrage

BIMQL ist auf die Extraktion von Teilmodellen ausgerichtet. Dadurch fehlen der Anfragesprache Funktionen für die Analyse von Gebäudemodellen. Wie in PMQL existieren keine Operatoren für die Erstellung des kartesischen Produkts aus Entitätsmengen und die Ausführung von Joins. Das Konzept der query shortcuts für die Selektion von Eigenschaften aus einer Eigenschaftsmenge zeigt hingegen, wie eine domänenspezifische Anfragesprache vorteilhaft auf spezielle Anforderungen und Gegebenheiten angepasst werden kann.

4.8.3 BERA

Building Environment Rule and Analysis (BERA) ist eine domänenspezifische Sprache für die Überprüfung von Gebäudemodellen gegenüber Designrichtlinien. Sie wurde im Rahmen einer Dissertation am *Georgia Institute of Technology* entwickelt (Lee, 2010). Die Sprache orientiert sich an ECMAScript und davon abgeleiteten Sprachen wie JavaScript (Wirfs-Brock & Terlson, 2009).

Designrichtlinien können sich aus projektspezifischen Anforderungen und gesetzlichen Vorschriften ergeben. Insbesondere wird im Falle von BERA auf amerikanische Richtlinien wie dem *Hospital Design Guide* und dem *U.S. Courts Design Guide* eingegangen. Die Hauptanwendungsgebiete der Sprache sind die Validierung von Evakuierungsstrategien und von Raumprogrammen. BERA nutzt hierzu ein eigenes Objektmodell, das für die genannten Anwendungsfälle ausgelegt ist. Dieses ist in Abbildung 4.10 abgebildet.

Anmerkung zu Abbildung 4.10: Hier wurden anscheinend diverse UML-Regeln nicht eingehalten. So werden die verwendeten Pfeile in umgekehrter Richtung eingesetzt. Zudem werden Vererbung und räumliches Enthaltensein mit der identischen Pfeilart ausgedrückt. Zur Repräsentation von Vererbungsbeziehungen ist dies jedoch die falsche Pfeilart und räumliche Beziehungen können in UML nicht allein durch Pfeile abgebildet werden.

Auf Basis der gezeigten Klassen werden das Objektmodell der Sprache und Konstrukte für die Prüfung von Designrichtlinien entwickelt (engl. *rule checking*). Auflistung 4.37

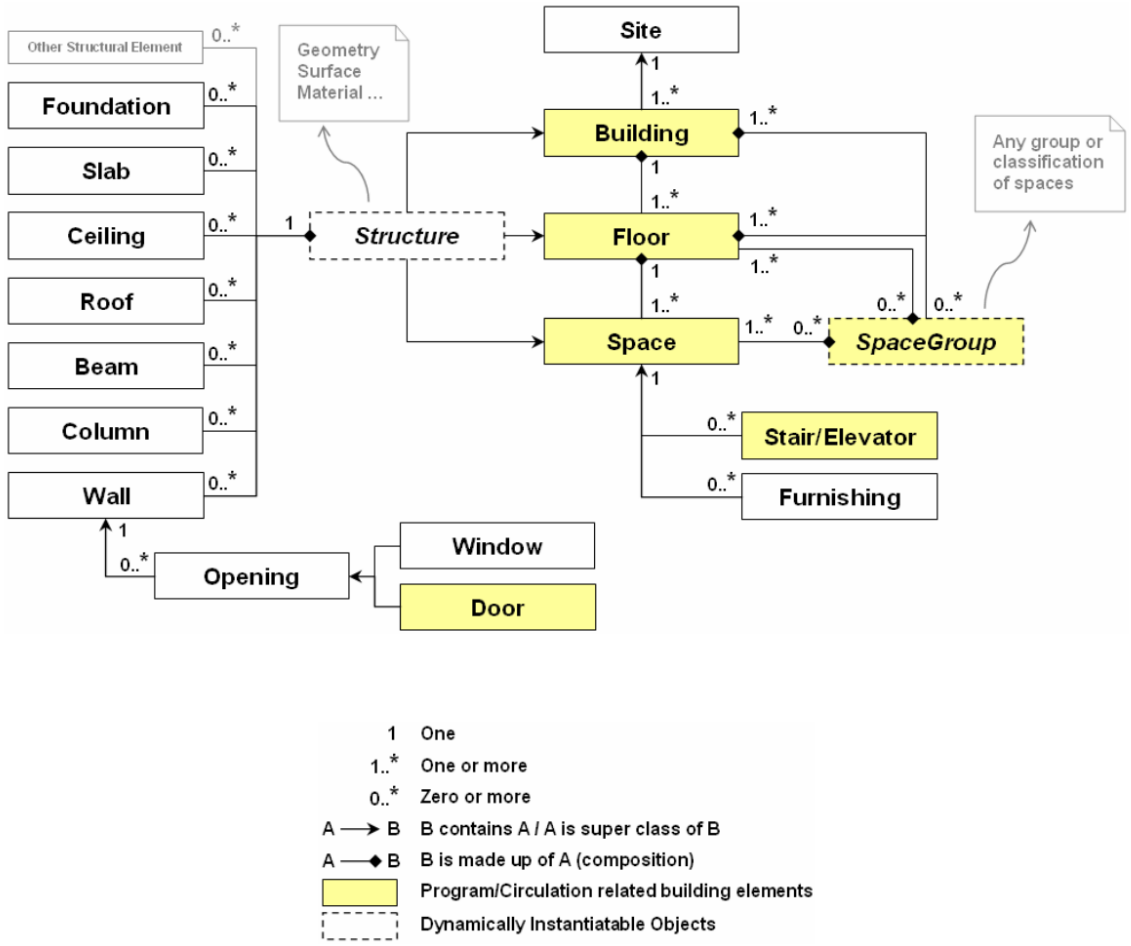


Abbildung 4.10: "An Abstraction of the Building Model for BERA Users (BERA-centered)", **unverändert übernommen** aus (Lee, 2010, S. 40)

zeigt ein BERA-Programm zur Untersuchung von Erreichbarkeiten zwischen dem `visitors conference`-Raum und den anderen Räumen des ersten Stockwerks. Zur Interaktion mit dem Gebäudemodell und zur initialen Extraktion von Entitäten für die anstehende Überprüfung werden Methoden genutzt. Anstelle einer parametrisierten Methode für unterschiedliche Typen wird für jeden Bauteiltyp eine explizite Variante bereitgestellt. Die Ergebnismenge ist jedoch über ein Prädikat weiter einschränkbar. In Zeile 1 werden so mit `getSpace(Space.Floor.number = 1)` alle Räume des ersten Stockwerks ausgewählt. Für das rule checking können Regeln genutzt werden, die aus mehren Prädikaten bestehen, Eingangsparameter entgegennehmen und Instanzen des Objektmodells überprüfen. Entsprechend wird die Regel `distanceRule` in den Zeilen 2 bis 6 definiert und in Zeile 8 mit dem Raumbezeichner `visitors conference` aufgerufen. Der Rest des Programms besteht aus Java-Code. Dieser wird benötigt um weitere Eigenschaften des Erreichbarkeitsgraphen zu berechnen. Dazu gehören metrische Distanzen und Abzweigungstiefen.

```
1  Space allGroundSpaces = getSpace(Space.Floor.number = 1);
2  Rule distanceRule(Space start, Space target) {
3    Path path = getPath(start, target);
4    path.distance < 100;
5    path.depth < 5;
6  }
7  distanceRule("visitors conference", allGroundSpaces);
8
9  java;
10 Double maxDistance = 0.0;
11 int maxTurn = 0;
12 int maxDepth = 0;
13 Space end1 = null;
14 Space end2 = null;
15 Space end3 = null;
16
17 for (Path p : Path) {
18   if (p.distance >= maxDistance) {
19     maxDistance = p.distance;
20     end1 = p.SpaceEnd;
21   }
22   if (p.numberOfTurn >= maxTurn) {
23     maxTurn = p.numberOfTurn;
24     end2 = p.SpaceEnd;
25   }
26   if (p.depth >= maxDepth) {
27     maxDepth = p.depth;
28     end3 = p.SpaceEnd;
29   }
30 }
31 [print statements removed]
```

Auflistung 4.37: Lee.2011b]BERA-Anweisungen zur Erreichbarkeitsanalyse zwischen Räumen, aus (Lee, 2010, S. 149)

BERA ist die Erweiterung einer Skriptsprache und zur Überprüfung von Designrichtlinien ausgelegt. Dies spiegelt sich im vereinfachten Objektmodell und in speziellen vorimplementierten Funktionen wider. Einen generellen Ansatz zur Analyse von IFC-Gebäudemodellen liefert BERA nicht, zumal allgemeine Operationen wie beispielsweise Joins und die Selektion nach Typzugehörigkeit fehlen. Für eine erweiterte Überprüfung von Richtlinien wird zudem Java in das Konzept integriert. Somit kann der Endnutzer nur auf einem speziellen, eingeschränkten Objektmodell agieren und muss neben BERA auch Java beherrschen. Obwohl die Sprache explizit für die Analyse von

räumlichen Aspekten des Modells ausgelegt ist, wird auf die Algorithmen zur räumliche Analyse in den vorhandenen Veröffentlichungen nicht detailliert eingegangen.

4.8.4 BIMcraft

BIMcraft ist ein Projekt, um eine modellzentrierte Arbeitsweise auch in der Ausführungsphase eines Bauprojektes fortzuführen (Wülfing et al., 2015). Dazu wird eine visuelle Sprache genutzt, um Informationsextraktion durchzuführen. Das fokussierte Anwendungsszenario ist die Angebotserstellung von Handwerkern unterschiedlicher Gewerke. Es wird davon ausgegangen, dass dabei mehrere Informationsquellen in einem gemeinsamen Kontext zusammengeführt werden müssen. So ermöglicht BIMcraft die kombinierte Analyse eines Gebäudemodells mit der Bauablaufplanung und entsprechenden Leistungsverzeichnissen. BIMcraft wurde zeitgleich mit den in dieser Arbeit gezeigten Vorgehen entwickelt und soll ebenfalls Operatoren zur räumlichen, zeitlichen und relationalen Filterung von Gebäudemodellen bereitstellen. Abbildung 4.11 zeigt eine visuelle BIMcraft-Anweisung zur Selektion von Räumen im ersten Stock mit einer Mindestgröße von 13 Quadratmetern.

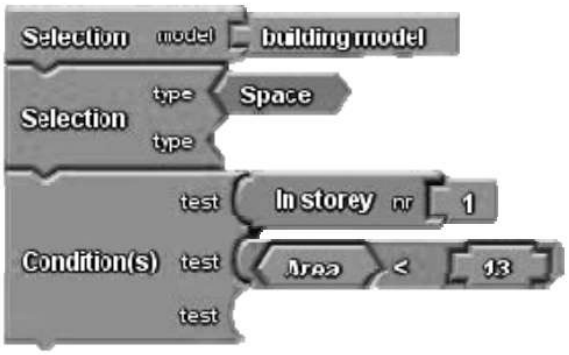


Abbildung 4.11: Beispielhafte BIMcraft Anfrage zur Selektion von Räumen, (Wülfing et al., 2015)

In BIMcraft wird auf die Einführung von Variablen verzichtet. So werden nachfolgende Operatoren stets auf die gesamte Ergebnismenge ausgeführt was die Funktionalität der Anfragen stark einschränkt. Außerdem steht keine textuelle Syntax für Anfragen bereit. Diese kann jedoch, beispielsweise für Client/Server-Umgebungen, nötig sein. Zudem wird in der einzig verfügbaren Veröffentlichung zu BIMcraft das verwendete interne Objektmodell nicht spezifiziert. Details zum Aufbau der Sprache, zur Ablaufumgebung und zu den Algorithmen der angedachten Operatoren werden nicht erörtert.

4.9 Fazit

Das EXPRESS-basierte IFC-Schema stellt eine objektorientierte Modellierung zur Repräsentation von Gebäudemodellen dar. So ermöglichen die IFC die vielfältigen Aspekte derartiger Modelle vorzuhalten. Dazu wird von einer hohen Anzahl an Entitäten in einer komplexen Vererbungshierarchie Gebrauch gemacht. In diesem Kapitel wurde die Eignung von allgemeinen Anfragesprachen zur Analyse von IFC-Gebäudemodellen anhand von Musteranfragen erörtert. Auch auf domänenspezifische Ansätze wurde eingegangen.

Abgesehen von XQuery erfordert der Einsatz von SQL-, LINQ, SPARQL-basierten Systemen eine Transformation des EXPRESS-IFC-Schemas zu dem der jeweiligen Anfragesprache. Eine derartige Transformation ist ohne eine verbindliche Spezifizierung problematisch. So ergeben sich divergierende Anfragedialekte, was die Übertragbarkeit von Analyse auf unterschiedliche Systeme bei der Nutzung von SQL-, LINQ und SPARQL einschränkt. Für den Endnutzer bedeutet die Verwendung eines von der ursprünglichen EXPRESS-Modellierung abweichenden Anfragemodells außerdem eine zusätzliche Anforderung. Im relationalen Modell muss beispielsweise die Atomarität von Attributen und die damit einhergehende Einführung von Containerrelationen korrekt eingesetzt werden.

Das IFC-Modell stellt neben standardmäßigen Attributen auch inverse Attribute bereit. Diese dienen dazu in Relationen die Richtung der Entitätstraversierung wählen zu können. In Anfragen zeigt sich, dass inverse Attribute die Anfragedefiniton aus Sicht des Fachexperten erleichtern können. Folglich kann dadurch beispielsweise von Wänden zu den beherbergten Bauteilen navigiert werden. So müssen als Ausgangspunkt keine objektorientierten Relationen gewählt und die Teilergebnisse durch Join-Operationen verknüpft werden. Daraus folgt, dass eine direkte Traversierung von Entitätsreferenzen und die Unterstützung inverser Attribute in IFC-Analysen vorteilhaft sind.

In Bezug auf die komplexe Vererbungshierarchie des IFC-Modells, wird die unzureichende Unterstützung für entsprechenden Typprüfungen in allen Systemen sichtbar. Die ausgereifteste Unterstützung von Vererbungsanalysen stellt dabei LINQ bereit. Die Sprache erfordert jedoch Cast-Operationen, wenn auf die Attribute von Subklassen zugegriffen werden soll. Umständliches Casten wird in den betrachteten Anfragesprachen zudem benötigt, wenn bestimmte Typen, beispielsweise Datumswerten verarbeitet werden sollen.

Für alle vorgestellten Anfragesprachen kann zusammenfassend festgestellt werden, dass bereits relativ einfache Musteranfragen zu komplexen Definitionen mit einer Vielzahl an syntaktischen und konzeptionellen Feinheiten führen. Daraus ergibt sich der Bedarf für eine domänenspezifische, an Fachanwender gerichtete Anfragesprache für 4D-Bauwerksmodelle.

Kapitel 5

Entwicklung domänenspezifischer Operatoren

In den Kapiteln 2 und 3 wurde auf Modellierungskonzepte der Geo- und Bauinformatik und speziell auf Stadt- und Gebäudemodelle eingegangen. Durch eine Untersuchung auf Basis von Musteranfragen konnten im letzten Kapitel Defizite identifiziert werden, wenn für die Analyse von Gebäudemodellen verbreitete Anfragetechnologien wie SQL, XQuery oder SPARQL eingesetzt werden. Auch die verfügbaren BIM-spezifischen Ansätze erlauben keine feingranularen Analysen auf gleichzeitig hohem Abstraktionsniveau. Abgesehen von einem passenden sprachbasierten Ansatz fehlen für derartige Analysen domänenspezifische Operatoren. Diese werden im Folgenden entwickelt. Wenn verfügbar werden diesbezüglich bestehende Definitionen verwendet.

Das Kapitel beginnt mit der Diskussion von Operatoren für die Untersuchung aufgabenbezogener zeitlicher Modellaspekte. Danach werden Operatoren für eine dreidimensionale räumliche Analyse von BIM- und GIS-Daten vorgestellt. Wegen der Rechenintensität derartiger Analysen stehen hier insbesondere effiziente Algorithmen im Vordergrund. Das Kapitel endet mit der Betrachtung von Operatoren für ein erweitertes Konzept zur Extraktion und Reintegration von Teilmodellen.

5.1 Zeitliche Operatoren

In Kapitel 3.5 wurde im Kontext der Terminplanung die Vorhaltung zeitlicher Daten und deren Integration in ein Gebäudemodell diskutiert. Wie besprochen liegen aufgabenbezogene zeitliche Daten in einer IFC-basierten Modellierung als IfcTask-Instanzen vor. Beziehungen zwischen vor- und nachgelagerten Aufgaben werden über IfcRelSequence vorgehalten. Aufgaben können zudem über IfcRelNests in Unteraufgaben gegliedert werden. Neben der Auswertung

dieser sequenziellen und hierarchischen Ordnung von Aufgaben ist eine zeitliche Betrachtung nötig. In dieser müssen explizite zeitliche Werte verarbeitet werden.

Gegenüber der Analyse des Bauablaufs mit konkreten Zeitwerten kann durch einen topologischen Ansatz eine Auswertung auf höherem Abstraktionsniveau realisiert werden. Die zeit-topologische Betrachtung von Geschehnissen (engl. occurrences) und die dafür nötigen mathematischen Grundlagen wurden unter anderem von Vilain (1982) untersucht, der 26 Prädikate zur Analyse von Zeitpunkten und Zeitintervallen vorstellt. Allen (1984) beschränkt sich hingegen in der *General Theory of Action and Time* auf die Betrachtung von Zeitintervallen.

Das primäre Modellierungselement der baulichen Terminplanung ist ebenfalls das Zeitintervall. Daher werden die 13 topologischen Prädikate von Allen als Ausgangspunkt der Entwicklung genutzt. Diese sind in Abbildung 5.1 dargestellt.

Relation	Symbol	Symbol for inverse	Pictoral example
X before Y	$<$	$>$	
X equal Y	$=$	$=$	
X meets Y	m	mi	
X overlaps Y	o	oi	
X during Y	d	di	
X starts Y	s	si	
X finishes Y	f	fi	

Tabelle 5.1: Die 13 unterscheidbaren Relationen zwischen Zeitintervallen, nach (Allen, 1984)

Für eine Analyse aufgabenbezogener zeitlicher BIM-Daten wird die Granularität der Operatoren jedoch als zu fein eingestuft. So ist zum Beispiel die Identifikation von Tätigkeiten interessant die sich zeitlich – ganz allgemein betrachtet – überlagern. Sind ausschließlich die Allen'schen Operatoren verfügbar muss sowohl auf `overlaps`, `during`, `starts`, `equal` und deren inverse Prädikate hin geprüft werden.

Um dies zu umgehen werden relaxierte Operatoren entwickelt. Diese weisen teilweise *kommutative* Eigenschaften auf und aggregieren mehrere Prädikate von Allen. Viele Fragestellungen können somit bereits mit `Before`, `After` und `Overlays` beantwortet werden. Während die

ersten beiden Operatoren vor- bzw. nachgelagerte Tätigkeiten identifizieren, selektiert der `Overlays`-Operator Zeitintervalle die sich überlagern. Im Gegensatz zu den Allen'schen Operatoren sind diese BIM-spezifischen Operatoren nicht eineindeutig und es kann beispielsweise `Overlays` und `Starts` gleichzeitig zutreffen. Dieses Verhalten ist beabsichtigt und ermöglicht sowohl gröbere als auch detaillierte Analysen ohne die Nutzung einer Vielzahl an Operatoren zu erzwingen. Um Überschneidungen mit den räumlich-topologischen Operatoren der Anfragesprache zu vermeiden, wurden die Bezeichnungen der zeitlichen Operatoren teilweise angepasst.

Die in QL4BIM verfügbaren Operatoren für die Analyse von Zeitintervallen sind in Abbildung 5.1 dargestellt. Tabelle 5.2 zeigt die mathematischen Definitionen der Operatoren. Aus diesen lässt sich die entsprechende Implementierung direkt ableiten.

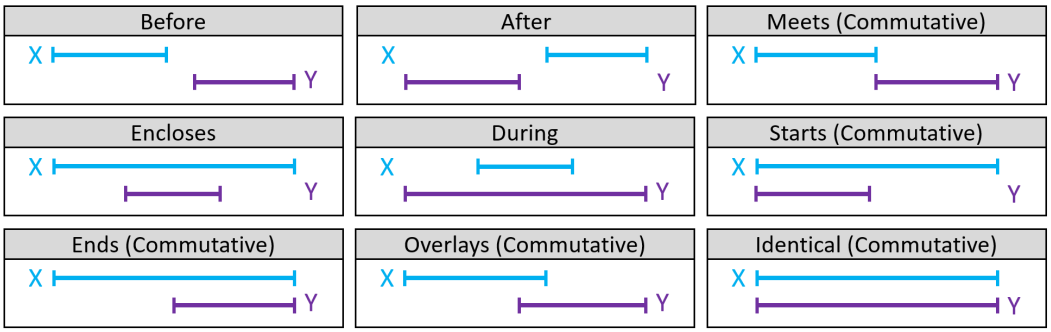


Abbildung 5.1: QL4BIM-Operatoren für die Analyse von Zeitintervallen innerhalb des Bauablaufs

Operator	Definition
X Before Y	$X_{max} < Y_{min}$
X After Y	$X_{min} > Y_{max}$
X Meets Y	$X_{max} = Y_{min} \vee X_{min} = Y_{max}$
X Encloses Y	$X_{min} < Y_{min} \wedge X_{max} > Y_{max}$
X During Y	$X_{min} > Y_{min} \wedge X_{max} < Y_{max}$
X Starts Y	$X_{min} = Y_{min}$
X Ends Y	$X_{max} = Y_{max}$
X Overlays Y	$X_{max} > Y_{min} \wedge X_{min} < Y_{max}$
X Identical Y	$X_{min} = Y_{min} \wedge X_{max} = Y_{max}$

Tabelle 5.2: Definitionen der zeitlichen QL4BIM-Operatoren

Die entwickelten zeitlichen Operatoren ermöglichen die Analyse des Bauablaufs auf Basis der topologischen Beziehungen zwischen Zeitintervallen. Dabei wurden die ursprünglich 13 Operatoren von Allen auf neun reduziert und so verändert, dass sowohl grobe als auch feingranulare zeitliche Untersuchungen möglich werden. Die zeitlich-topologischen Operateur lassen sich mit Attributauswertungen und Dereferenzierungs-

operatoren kombinieren, wodurch vielfältige Fragestellungen im Kontext einer 4D-Modellierung beantwortet werden können. Darauf wird in Kapitel 8.1.1 näher eingegangen. Empirische Tests zeigten außerdem, dass eine Indizierung der Zeitintervalle nicht erforderlich ist, da sich die ergebenden Laufzeiten im Bereich der nichträumlichen Operatoren bewegen.

5.2 Räumliche Operatoren

Die semantische Analyse von BIM-Daten basiert primär auf der Auswertung von Entitätsattributen und der Typzugehörigkeit von Entitäten. Derartige Funktionalität lässt sich mit einer Hochsprache wie Java oder C# direkt umsetzen und in ein entsprechendes Laufzeitsystem der Abfragesprache integrieren. Dies ist für die räumlichen Operatoren nicht der Fall. So sind teilweise keine passenden algorithmischen Konzepte verfügbar. Deren Entwicklung ist Thema des folgenden Abschnitts. Zusätzlich müssen die erzielbaren Ausführungsgeschwindigkeiten betrachtet werden, da die räumlichen Operatoren mit den aufwendigsten Berechnungen behaftet sind, gleichzeitig aber ein interaktives BIM-Analysesystem angestrebt wird. Im Rahmen dieser Arbeit wird daher für die räumlichen Operatoren eine Indizierung mit R*-Trees vorgeschlagen.

5.2.1 Verwandte Arbeiten

Wie in Kapitel 2.4.2 beschrieben, ist eine formale Definition topologischer Relationen auf Basis der Punktmengentopologie verfügbar. Dies umfasst das 4-IM, in dem die Schnittmengen zwischen Innerem und Rand zweier räumlicher Entitäten untersucht und mit Vorlagen verglichen werden. In der 9-IM wird zudem das Äußere der Entitäten mit in die Untersuchung einbezogen.

Im Bereich der *Computational Geometry (CG)*, einschließlich der Transferformate IFC und CityGML, werden räumliche Entitäten primär durch prozedurale und BRep-basierte Beschreibungen vorgehalten. Prozedurale Geometrie lässt sich in BRep überführen, wodurch eine einheitliche Geometrierepräsentation für alle unterstützten Formate verfügbar ist. Auf Grund der impliziten Innen/Außen-Klassifizierung ist jedoch eine 4-IM- bzw. 9-IM-Matrix aus zwei BReps nicht direkt ableitbar.

Um dennoch **topologische Prädikate** zwischen räumlichen BIM-Entitäten ableiten zu können, wird in (Borrmann & Rank, 2009b) eine geometrische Zwischenrepräsentation genutzt. Konkret werden *Octree*-Datenstrukturen eingesetzt, deren Oktanten nach ihrer relativen Lage zum ursprünglichen BRep unterschiedlich eingefärbt werden. Äußeren Oktanten wird die Far-

be Weiß, auf dem Rand liegenden, Grau und inneren, Schwarz zugewiesen. Somit stehen die zwei zu untersuchenden räumlichen Entitäten als dreifarbige Oktalbäume zur Verfügung. Da die Bäume den identischen räumlichen Bereich abdecken, können überlagernde Oktanten verglichen werden. Dazu wird eine parallele Baumtraversierung ausgeführt, in der die Oktalbäume verfeinert und Farbkombinationen der Oktanten registriert werden. Die Baumverfeinerung und Farbauswertung wird so lange durchgeführt bis das zu untersuchende Prädikat bestätigt oder widerlegt werden kann. Methodisch entspricht dies der Erstellung einer minimal besetzen 9-IM-Matrix auf Basis einer zusätzlichen, diskretisierten Geometrierepräsentation.

Borrmann & Rank (2009a) untersuchen zudem die verfügbaren Definitionen für **direktionale Prädikate**. Die Diskussion umfasst kegel-, projektions- und intervallbasierte Ansätze (Frank, 1996; Guesgen, 1989; Goyal, 2000). Keiner der Ansätze wird für die räumliche Analyse von BIM-Daten als geeignet eingestuft. Der Grund sind die angewendeten Approximationen und zu feingranulare Prädikate. Daher wurden neue Definitionen speziell für dreidimensionale Körper entwickelt.

Für die algorithmische Umsetzung der so definierten direktionalen Prädikate wird von Oktalbäumen ausgehend eine weitere Datenstruktur abgeleitet. Ein *Slot* stellt einen direktional betrachten Schnitt innerhalb eines Oktalbaums dar. Wie Oktalbäume lassen sich auch Slots überlagern, wodurch die Berechnung direktionaler Prädikate ermöglicht wird. Abbildung 5.2 illustriert das oktalbaumbasierte Verfahren zur Untersuchung topologischer Prädikate und zeigt eine beispielhafte Slot-Datenstruktur.

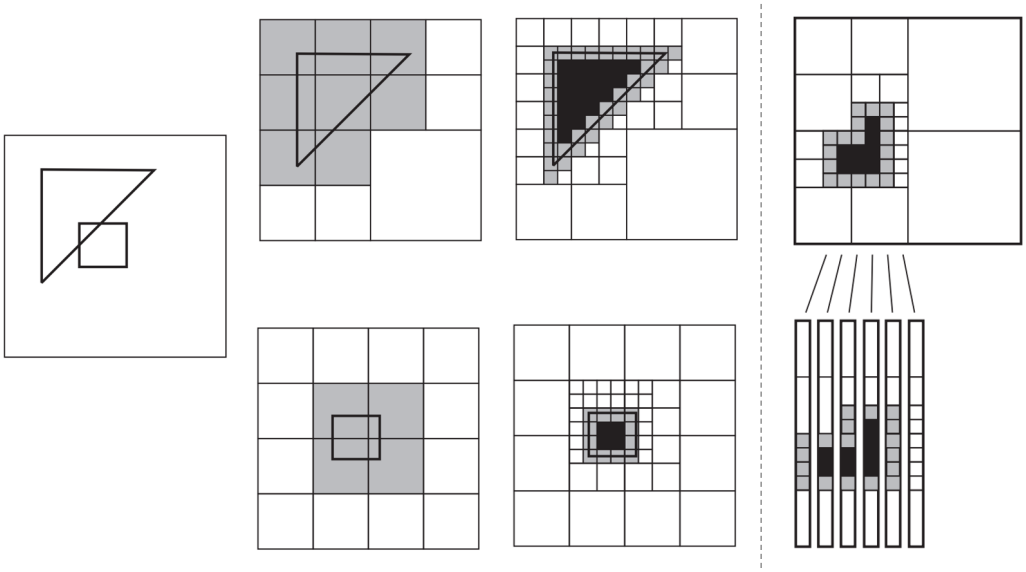


Abbildung 5.2: Links: Bestimmung topologischer Prädikate mit Hilfe von diskretisierten Geometrierepräsentationen; rechts: beispielhafte Slot-Datenstruktur, Darstellung jeweils in 2D, aus (Borrmann & Rank, 2009a, S. 9; Borrmann & Rank, 2009b, S. 12)

Die octree- und slotbasierten Ansätze realisieren eine topologische und direktionale Auswertung dreidimensionaler räumlicher Entitäten, weisen jedoch zugleich die im Folgenden beschriebenen Nachteile auf.

- Die Genauigkeit der Auswertungen entspricht der minimalen Oktantengröße. Bereits bei moderaten Genauigkeitsanforderungen im Zentimeterbereich müssen bei ungünstigen räumlichen Konstellationen somit tiefe Oktalbäume generiert werden. Deren Erstellung erfordert aufwendige Berechnungen und erzeugt speicherintensive Strukturen.
- Die Auswertung derartiger Oktalbäume und daraus abgeleiteter Slot-Strukturen führen zu Ausführungszeiten die für ein interaktives Analysesystems in Kombination mit datenintensiven Gebäudemodellen als nicht praktikabel einzuschätzen sind (Borrmann & Rank, 2009b, S. 16).
- Das Ergebnis der topologischen Auswertung ist nicht nur von der räumlichen Konstellation der untersuchten Entitäten abhängig sondern auch von der gewählten Ausdehnung der Oktalbäume. Somit kann das aufgezeigte Verfahren falsche Ergebnisse liefern, die nur durch Änderung der Baumausdehnung, einer Baumneugenerierung und einer erneuten Auswertung erkannt werden können. Diese ist mit einer zusätzlichen Erhöhung der Ausführungszeit verbunden.

Daher wird im Zuge dieser Arbeit eine neue Methode für die Untersuchung topologischer und direktionaler Relationen zwischen dreidimensionalen Körpern entwickelt. Die Methode nutzt keine zusätzliche Geometrierepräsentation, sondern arbeitet direkt auf dreiecksbasierter Geometrie. Auch in diesem Ansatz kommen Baumstrukturen zum Einsatz. Diese dienen zur Indizierung räumlicher Elemente und verringern die Zeitkomplexität der entwickelten Algorithmen.

5.2.2 Grundlegende BRep-Algorithmen

Das entwickelte Verfahren für die effiziente Bestimmung topologischer Prädikate zwischen dreidimensionalen Körpern basiert auf der Verarbeitung *triangulierter* BRep-Strukturen. Diese können sowohl aus prozeduralen sowie polygonbasierten Beschreibungen, respektive aus IFC- und CityGML-Daten, gewonnen werden. Für beide Beschreibungsarten existieren ausgereifte Konvertierungsverfahren, zumal eine hardwarebeschleunigte 3D-Visualisierung ebenfalls dreiecksbasiert arbeitet (Shreiner, 2013, Kap. 1).

Nach der Konvertierung der IFC- und CityGML-spezifischen Geometriebeschreibungen werden die gewonnenen, triangulierten BReps durch das Laufzeitsystem der Anfragesprache vorgehalten und mit den entsprechenden semantischen Entitäten verknüpft. Die ursprünglichen Geometriebeschreibungen der Entitäten werden für den Exportfall beibehalten.

Die Egenhofer'schen Definitionen erfordern eine Unterscheidung zwischen Innerem, Rand und Äußerem einer räumlichen Entität. Daher müssen die genutzten, triangulierten Flächenmodelle geschlossene Körper, sogenannte *solid*s bzw. *shell*s, darstellen (ISO, 1992; OGC, 2012, Kap. 8.1; IfcSolidModel in Tabelle 3.3). In CityGML wird die Modellierung geschlossener Körper im Gegensatz zur IFC nicht erzwungen. Es existieren jedoch Ansätze, geschlossene Körper aus einzelnen Flächenstrukturen abzuleiten (FME, 2016; OGC, 2012, Kap. 6.4).

In den nachfolgenden Algorithmen wird außerdem von BRep-Strukturen mit einer einfachen, nicht verschachtelten Hüllgeometrie ausgegangen. Sollen Körper mit innen liegenden Aussparungen, sogenannten *caves* unterstützt werden, müssen algorithmische, ggf. rekursive Erweiterungen entwickelt werden.

BRep-basierte topologische Operatoren

Linien- und Flächenobjekte besitzen in BIM-Daten zumeist annotierenden Charakter und stellen nur in Ausnahmefällen auszuwertende Geometrie dar. Daher werden im Folgenden ausschließlich geschlossene, dreidimensionale Körper betrachtet. Diese verhalten sich topologisch wie Regionen im zweidimensionalen Raum (Egenhofer & Herring, 1994; Borrmann & Rank, 2009b). Somit kann zur Relationsbestimmung anstelle des 9-IM das einfachere 4-IM genutzt werden.

Die in dieser Arbeit entwickelte Topologieanalyse basiert auf **Schnitt-** und **Berührtests** zwischen Dreiecken sowie der **Innen/Außen-Klassifizierung** von Dreiecken. Durch unterschiedliche Kombinationen dieser Untersuchungen ist es möglich, ein eindeutiges 4-IM zu gewinnen und somit ein topologisches Prädikat zwischen einem Entitätspaar zu bestätigen oder zu widerlegen.

Ein effizienter **Dreiecksschnitt-Test** auf Basis von Segmentprojektion wurde von Möller (1997) entwickelt. T_1 und T_2 seien zwei Dreiecke und π_1 und π_2 die durch sie definierten Ebenen. Als erstes werden die vorzeichenbehafteten Distanzen zwischen allen *Vertices* von T_1 und Ebene π_2 berechnet. Weisen alle Distanzen das identische Vorzeichen auf, können sich die Dreiecke nicht schneiden und der Test kann abgebrochen werden. Dieser Vortest wird für Dreieck T_2 und Ebene π_1 wiederholt. Falls kein Abbruch erfolgt, wird die Schnittgerade L zwischen Ebene π_1 und Ebene π_2 berechnet. Durch die Vortests ist sichergestellt, dass beide Dreiecke Intervalle auf L bedecken. Die Dreiecke schneiden sich nur dann, wenn sich diese Intervalle überlagern.

Zur Untersuchung von topologischen Relationen muss der Algorithmus erweitert werden, um Schnitt- und Berührfälle zwischen Dreiecken gesondert behandeln zu können. Stellt die Überlagerung ein Intervall dar, schneiden sich T_1 und T_2. Tritt eine punktuelle Interaktion auf, berühren sich die beiden Dreiecke. Die numerische Unterscheidung zwischen Punkt- und Intervall-

Interpretation erfolgt durch einen *Epsilon*-Wert. Abbildung 5.3 zeigt ein Beispiel zweier sich schneidender Dreiecke, die Schnittgerade zwischen π_1 und π_2 und das Intervall der Überlappung. Außerdem wird die Unterscheidung zwischen Schnitt- und Berührfällen illustriert.

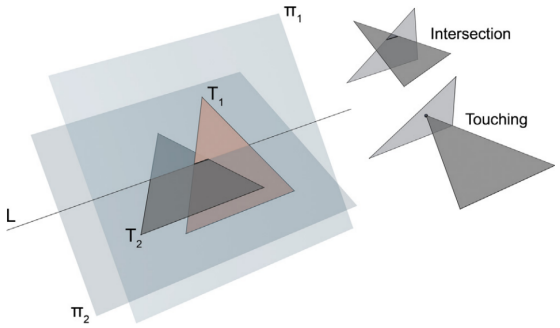


Abbildung 5.3: Schnitt- und Berührtests zwischen Dreiecken auf Basis von Segmentprojektionen, nach (Möller, 1997).

Die **Innen/Außen-Klassifizierung** wird über die Ray-Intersection-Methode umgesetzt, die in Kapitel 2.4.1 in Bezug auf den Punkt-in-Polygon-Test besprochen wurde. In dieser 3D-Variante wird die Relation zwischen zwei BRep-Strukturen untersucht. Anstelle von Strahl-Kanten-Schnitten werden hier Strahl-Dreieck-Schnitte gezählt. Als Vorbedingung der Klassifizierung muss sichergestellt werden, dass keine Dreiecksschnitte zwischen den betrachteten BReps vorliegen.

Zur Klassifizierung eines BReps muss ein zugehöriger Punkt als Emissionsstart ausgewählt werden. Dies geschieht über die Selektion eines geeigneten Dreiecks des betrachteten BReps. Dreiecke einer Entität, die Dreiecke der zweiten Entität berühren, dürfen nicht als Ausgangsflächen für Strahlen gewählt werden, da sonst fehlerhafte Klassifizierungen erfolgen können. Aus Gründen der Verständlichkeit wird dies im Pseudocode des Kapitels nicht berücksichtigt.

Mit Hilfe der vorgestellten Schnitt- und Berührtests zwischen Dreiecken sowie der Innen/Außen-Klassifizierung von Dreiecken können drei Hilfsfunktionen `Intersects`, `Meets` und `IsInside` eingeführt werden. Auflistung 5.1 zeigt die Signaturen der drei Funktionen.

```
bool Intersects(Triangle A, Triangle B)
bool Meets(Triangle A, Triangle B)
bool IsInside(Triangle TriangleA, BRep BRepB)
```

Auflistung 5.1: Hilfsfunktionen zur Untersuchung von Dreiecksschnitten und -berührungen, sowie zur Innen/Außen-Klassifizierung eines Dreieckes bezüglich eines BReps

Durch unterschiedliche Kombinationen der Hilfsfunktionen kann aus jeder der acht Vorlagematrizen des 4-IM ein BRep-verarbeitender Algorithmus erstellt werden. Für die `Contains`- und `Touches`-Prädikate wird hierauf näher eingegangen. Die entsprechenden 4-IM-Matrizen sind in Abbildung 5.4 dargestellt.

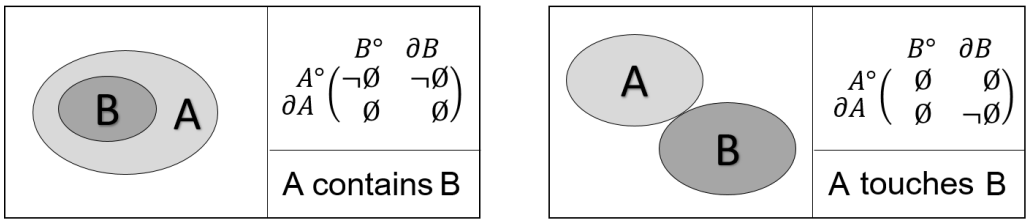


Abbildung 5.4: Die 4-IM-Matrizen der Contains- und Touches-Prädikate

Auflistung 5.2 enthält den Pseudocode zur Untersuchung des `Contains`-Prädikats. Zu Beginn wird eine verschachtelte Iteration über die Dreiecke von BRep A und B ausgeführt. Dabei wird untersucht, ob Dreieckspaare existieren, die sich schneiden oder berühren. Im Fall eines Schnitts ist $\partial A \cap B^\circ = \neg\emptyset \wedge \partial A \cap \partial B = \neg\emptyset$. Bei einer Berührung gilt $\partial A \cap \partial B = \neg\emptyset$. Dadurch schließen die Berührung und der Schnitt von Dreiecken das Prädikat aus. Tritt keine der beiden Interaktionsfälle auf, muss zur Bestätigung von `Contains` noch untersucht werden, ob sich B innerhalb von A befindet. Dann gilt $A^\circ \cap B^\circ = \neg\emptyset \wedge A^\circ \cap \partial B = \neg\emptyset$. Hierzu wird ein Dreieck von B bezüglich seiner relativen Lage zu A hin untersucht. Bestätigst sich das Enthaltensein, ist die Vorlagematrix vollständig abgeglichen und der Algorithmus liefert `true` zurück.

```
boolean Contains(BRep A, BRep B) //A contains B
  foreach triangleA in A
      foreach triangleB in B
        if Intersects(triangleA, triangleB) || Meets(triangleA, triangleB)
          return false
  return IsInside(B.Triangles[0], A)
```

Auflistung 5.2: BRep-basierter Algorithmus zur Bestimmung des Contains-Prädikats

Der `Touches`-Algorithmus ist in Auflistung 5.3 dargestellt. Zur Verifikation des Prädikats muss in der Dreiecksinteration mindestens eine Berührung auftreten. Dadurch gilt $\partial A \cap \partial B = \neg\emptyset$. Wird hingegen ein Dreiecksschnitt erkannt, folgt daraus $A^\circ \cap B^\circ = \neg\emptyset \wedge A^\circ \cap \partial B = \neg\emptyset \wedge \partial A \cap B^\circ = \neg\emptyset$. Die beiden letzten Teilaussagen widerlegt das Touches-Prädikat. Auch ohne das Auftreten eines Dreieckschnitts kann $A^\circ \cap B^\circ = \neg\emptyset$ zutreffen und zwar, wenn sich A in B oder B in A befindet. Sind diese beiden Fälle zusätzlich ausgeschlossen ist die Vorlagematrix des Touches-Prädikats vollständig überprüft und der Operator liefert `true` zurück.

```
boolean Touches(BRep A, BRep B) //A touches B
  meets = false
  foreach triangleA in A
      foreach triangleB in B
        if meets || Meets(triangleA, triangleB)
          meets = true
        if Intersects(triangleA, triangleB)
          return false
  if !meets
    return false
  return !IsInside(B.Triangles[0], A) && !IsInside(A.Triangles[0], B)
```

Auflistung 5.3: BRep-basierter Algorithmus zur Bestimmung des Touches-Prädikats

Das diskutierte Konzept ermöglicht die Berechnung der acht topologischen Prädikate des 9-IMs durch die direkte Verarbeitung von BRep-Geometrie. Für Entitäten mit einfachen geometrischen Repräsentationen und einer damit einhergehenden geringen Anzahl an Dreiecken ist somit bereits eine effiziente Methode zur topologischen Untersuchung verfügbar. Für Geometrierepräsentationen mit einer hohen Anzahl an Dreiecken muss das Skalierungsverhalten des Ansatzes verbessert werden. In Abschnitt 5.2.3 wird der diesbezügliche Einsatz von R*-Trees erörtert.

BRep-basierte direktionale Operatoren

In Borrmann & Rank (2009a) werden eine *halbraum-* und eine *projektionsbasierte* Definition für direktionale Operatoren eingeführt. Somit können Entitäten die übereinander oder untereinander liegen bzw. in den Himmelsrichtungen Norden, Süden, Osten und Westen miteinander interagieren, selektiert werden. In der projektionsbasierten Methode wird die Referenzentität A in die jeweils untersuchte Richtung, also stets parallel zu einer Koordinatenachse, extrudiert und auf Schnitte mit der Testentität B hin untersucht. Dies liefert aussagekräftigere Ergebnisse als das halbraumbasierte Vorgehen. Daher nutzen die hier vorgestellten direktionalen Operatoren die projektionsbasierte Definition. Diese umfasst eine strikte und eine relaxierte Variante.

A und B seien räumliche Körper und $a \in A, b \in B$. Für das Referenzobjekt A und das Testobjekt B lauten die Definitionen des Above-Prädikats wie folgt.
$aboverelaxed(A,B) \Leftrightarrow \exists a,b : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z$
$abovestrict(A,B) \Leftrightarrow \forall b : (\exists a : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z) \wedge (\nexists a : a_x = b_x \wedge a_y = b_y \wedge a_z \geq b_z)$
Die Definitionen der restlichen Operatoren Below, NorthOf, SouthOf, EastOf und WestOf sind in Borrmann & Rank (2009a) enthalten.

Anstelle der Auswertung von Slot-Strukturen wird in dieser Arbeit ein Verfahren beschrieben, das auf der direkteren Untersuchung direktionaler Dreiecksbeziehungen beruht. In der relaxierten Variante werden dazu Prismen-Dreieck-Schnitte, in der strikten Variante Strahl-Dreieck-Schnitte bzw. Dreiecksprojektionen eingesetzt.

Zur Bestätigung eines relaxierten direktionalen Prädikats reicht das Auftreten eines einzelnen Prismen-Dreieckschnitts aus. Über eine Bounding Box-Auswertung wird zu Beginn der Maximalwert von Testobjekt B in die zu untersuchende Richtung ermittelt. Im `Above`-Fall ist dies der positive Z-Wert. Anschließend werden aus den Dreiecken von Referenzobjekt A Prismen erzeugt, die sich bis zu diesem Z-Wert ausdehnen. Diese Prismen werden gegenüber den Dreiecken von Testobjekt B auf Schnitt hin untersucht. Zudem folgt aus der Definition, dass eine Überlappung der beiden Objekte ebenfalls zu einem positiven Ergebnis führt. Dieser Fall wird über den topologischen `Overlaps`-Operator behandelt. Abbildung 5.5 illustriert das Vorgehen für das `Above`-Prädikat. Der entsprechende Pseudocode ist in Auflistung 5.4 enthalten.

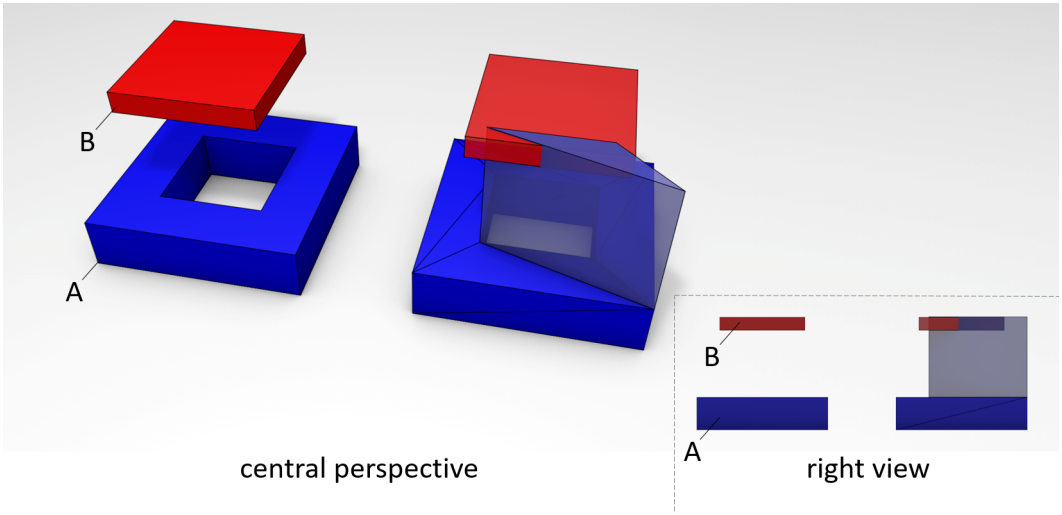


Abbildung 5.5: Prismenerstellung durch Extrusion und Prismen-Dreieck-Schnitttest zur Umsetzung der direktionalen Operatoren

```
boolean AboveRelaxed(BRep A, BRep B)
  if Overlaps(A,B)
    return true
  maxZ = B.Bounds.MaxZ
  foreach triA in A
    prismA = new Prisma(triA, maxZ, Direction.ZPositiv)
    foreach triB in B
      if Intersects(prismA, triB)
        return true
  return false
```

Auflistung 5.4: BRep-basierter Algorithmus zum Bestimmung relaxierter, direktionaler Prädikate (Above-Variante)

Sowohl Prisma als auch Dreieck stellen konvexe Körper dar. Für die Schnittuntersuchung zwischen konvexen Körpern existiert mit der *Method of Separating Axes* ein effizienter und robuster Ansatz (Eberly, 2001). Die Methode basiert auf der Erkenntnis, dass sich zwei konvexe Körper nicht schneiden können, wenn eine Achse existiert auf der sich die Projektionen der Körper nicht schneiden. Zwischen *polyhedralen* Körpern kommen nur bestimmte Achsen als Seperationsachsen in Frage. Zum einen sind dies die *Normalenvektoren* der beteiligten Dreiecke. Außerdem ergeben die *Kreuzprodukte* von Kanten beider Körper relevante Achsen. Für Prisma-Dreieck-Untersuchungen ergeben sich somit 17 Kandidatenachsen, die auf Intervallüberlagerungen geprüft werden müssen. Der Pseudocode für die entsprechende `Intersects`-Methode ist in Auflistung 5.5 dargestellt.

```
bool Intersects(Prism prismA, Triangle triB)
prismNormals = Crossproduct foreach prismA.Base.Edge with extrusion direction
 + prismA.Base.Normal
triNormals = Crossproduct foreach triB.Edge with triB.Normal + triB.Normal
cpAxes = Crossproduct foreach triB.Edge with prismA.Edge
foreach axes in (prismNormals + triNormals + cpAxes)
  intervalPrismA = Interval.Union(Project all prismA.Edges)
  intervalTriB = Interval.Union(Project all triB.Edges)
    if (!intervalPrismA.Intersects(intervalTriB))
        return false
return true
```

Auflistung 5.5: Schnitttest zwischen Prisma und Dreieck auf Basis der Method of Separating Axes

Für die Bestätigung des strikten `Above`-Prädikats reicht das Auftreten eines Prisma-Dreieck-Schnitts nicht aus. Stattdessen muss bewiesen werden, dass alle Dreiecke des Testobjekts B komplett über Dreiecken des Referenzobjekts A liegen.

Für die angedachte Anfragesprache werden zwei BRep-basierte Ansätze zur Untersuchung strikter direktionaler Prädikate vorgeschlagen. In einem *analytischen* Ansatz werden Dreiecke von A auf die entsprechende Ebene projiziert und zur Polygonen vereinigt. In einer Iteration über alle Dreiecke von B wird nun getestet, ob sich deren Projektionen innerhalb der A-Polygone befinden.

Der zweite Ansatz ist ein *heuristisches* Verfahren, in dem *Samplingpunkte* in jedem B-Dreieck verteilt werden. Von diesen Punkten werden Strahlen in die Gegenrichtung der Extrusion ausgesendet. Schneidet ein Strahl kein Dreieck von A, ist das direktionale Prädikat widerlegt.

Abbildung 5.6 zeigt eine beispielhafte Konstellation zweier räumlicher Körper, in denen `AboveStrict(A, B)` zutrifft. Der Pseudocode der heuristischen Variante ist in Auflistung 5.7 enthalten, der Code der analytischen Variante in Auflistung 5.6.

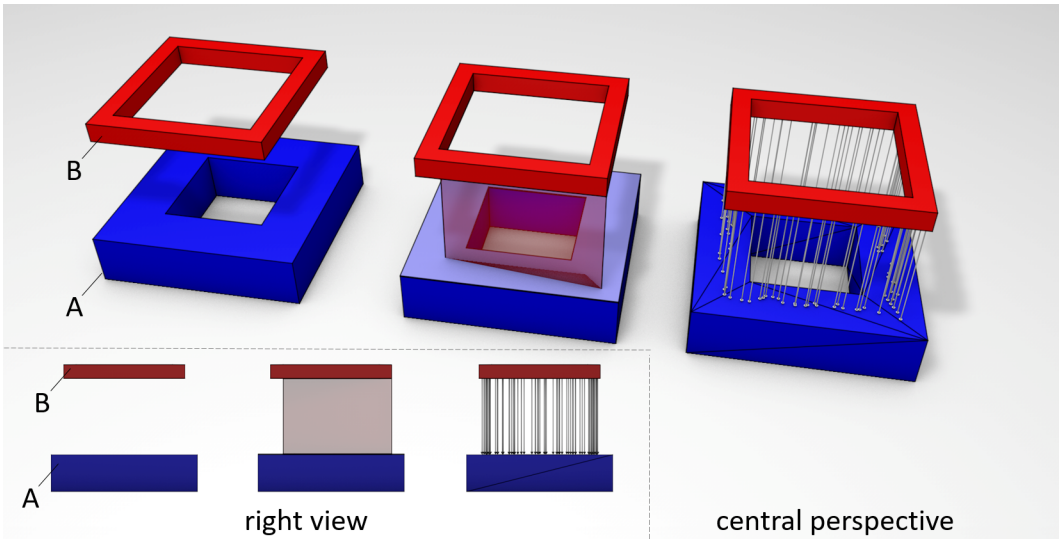


Abbildung 5.6: Strikte direktionale Prädikate lassen sich über Projektionen analytisch und über ein Samplingverfahren mit Strahlentests heuristisch lösen

```
boolean AboveStrictAnalytic(BRep A, BRep B)
  extendedBoxB = B.Bound.Extend(A.Bound, Direction.YNegative)
  if(!extendedBox.Intersect(A.Bound))
    return
  proPolysA = List
  foreach triangleA in A
    proPolysA.Add(triangleA.ProjectOnXY)
  foreach triangleB in B
    if(!proPolysA.InsideAny(triangleB.ProjectOnXY))
      return false
  return true
```

Auflistung 5.6: Analytischer BRep-basierter Algorithmus zum Bestimmung strikter, direktionaler Prädikate (Above-Fall)

```
boolean AboveStrictHeuristic(BRep A, BRep B)
  foreach triangleB in B
    triangleB.CreateSamplingPoints()
    foreach point in triangleB.SamplingPoints
      if(!A.IntersectAnyTriangle(new Ray(point, Direction.YNegative)))
        return false
  return true
```

Auflistung 5.7: Heuristischer BRep-basierter Algorithmus zur Bestimmung strikter, direktionaler Prädikate (Above-Fall)

Das analytische Verfahren erfordert deutlich geringeren Berechnungsaufwand als das heuristische Verfahren, insbesondere wenn eine hohen Samplingauflösungen genutzt wird. Die analytische Methode ist jedoch aufwendiger in der Implementierung, da hier die Erstellung komplexer Polygone mit Hohlräumen unterstützt werden muss. Beide Ansätze sind anfällig gegenüber Fehlinterpretationen aufgrund minimaler numerischer Abweichungen zwischen den untersuchten Geometrierepräsentationen. Um dem entgegenzuwirken wird in Abschnitt 5.2.4 ein toleranzunterstützender Algorithmus für das heuristische Verfahren beschrieben. Wie für die topologischen ist auch für die direktionalen Operatoren eine Indexierung sinnvoll, wenn detailreiche Geometrierepräsentationen untersucht werden müssen.

BRep-basierte metrische Operatoren

Die metrischen Operatoren `Nearer` und `Farther` dienen der Selektion von Entitäten durch Auswertung minimaler Abstände. Die grundlegende Definition der Operatoren kann aus (Borrmann & van Treeck, 2006) übernommen werden. Seien A und B räumliche Körper und $a \in A, b \in B$ dann gilt:

$Distance(A, B) := min_{a,b}(d(a, b))$
$Nearer(A, B, c) \Leftrightarrow min_{a,b}(d(a, b)) < c$
$Farther(A, B, c) \Leftrightarrow min_{a,b}(d(a, b)) > c$

Die `Distance`-Funktion ermittelt den minimalen Abstand zwischen den Körpern A und B. Im Falle, dass sich die Körper schneiden, liefert die Funktion den Wert 0 zurück. Das Argument c in den Operatoren `Nearer` und `Farther` ist eine reelle, positive Zahl und stellt eine durch den Endnutzer wählbare obere bzw. untere Grenze dar.

Zur Umsetzung der Operatoren auf Basis von BRep-Prozessierung wird ein Algorithmus zur minimalen Abstandstandsberechnung zwischen Dreiecken benötigt. Dieser baut auf der Abstandsberechnung zwischen Punkten und Dreiecken auf, wofür (Jones, 1995) und Eberly (1999) Verfahren beschreiben. Durch die somit umsetzbare Hilfsfunktion `Distance` können die metrischen Operatoren in einer ersten Version erstellt werden.

Auflistung 5.8 zeigt den Algorithmus für den `Nearer`-Operator. Für `Farther` ändert sich ausschließlich der Vergleichsoperator in Zeile 6.

```
1 boolean Nearer(BRep A, BRep B, double bound)
2   foreach triA in A
3     foreach triB in B
4       dist = Distance(triA, triB)
5         if dist < bound
6           return true
7 return false
```

Auflistung 5.8: BRep-basierter Algorithmus zur Bestimmung metrischer Prädikate (Nearer-Fall)

Wegen der quadratischen Zeitkomplexität des Verfahrens und der rechenintensiven Abstandsberechnung zwischen Dreiecken ist eine räumliche Indexierung erforderlich. Hierfür sind bereits erprobte Verfahren verfügbar.

5.2.3 Räumliche Indexierung

Die vorgestellten Algorithmen ermöglichen prinzipiell die Untersuchung topologischer, direktionaler und metrischer Prädikate durch Verarbeitung dreiecksbasierter Geometrierepräsentationen. Allerdings muss das Skalierungsverhalten der Algorithmen in Hinblick auf komplexe Geometrie kritisch betrachtet werden. Dazu wird n als die mittlere Dreiecksanzahl eines BReps in einem Gebäudemodell definiert. Durch die verschachtelten Dreiecksiterationen weisen die räumlichen Operatoren in ihrer derzeitigen Form eine Zeitkomplexität von $O(n^2)$ auf.

Zwar können AABB-Vortests zur Reduktion der auszuwertenden Entitäten genutzt werden. Kann die Interaktion zwischen Entitäten jedoch hierdurch nicht ausgeschlossen werden, müssen die kompletten BRep-Datenstrukturen ausgewertet werden. Um die Zeitkomplexität der räumlichen Operatoren zu verbessern, wird das Verfahren mit einer hierarchischen Indexstruktur ergänzt. Die Struktur ermöglicht es frühzeitig, eine Vielzahl an Kandidaten von weiteren Tests auszuschließen. In diesem Kontext stellen Dreieckspaare der beiden zu untersuchenden Entitäten diese Kandidaten dar. Somit können auch bei Entitäten deren AABBs interagieren, räumliche Prädikate früher ausgeschlossen bzw. bestätigt werden. Dies reduziert die Ausführungszeiten von Anfragen bei der Analyse datenintensiver Modelle.

Daher wird zur effizienten Auswertung räumlicher Relationen in BIM-Daten die R*-Tree-Datenstruktur vorgeschlagen (siehe Kapitel 2.5.3). Für jede Entität mit einer Geometrierepräsentation wird ein eigener Baum zur Indizierung der enthaltenen Dreiecke generiert. Die Indexstruktur wird in den räumlichen Algorithmen in zwei Varianten genutzt:

- Ein Suchbereich wird dem Baum **einer** Entität als Filterkriterium übergeben. Alle den Bereich schneidenden und enthalten Dreiecke werden zurückgegeben.

- Die Bäume **zweier** Entitäten werden in einer parallelen Tiefensuche traversiert. Nur die nötigen Knotenpaare beider Bäume werden bei voranschreitender Baumtiefe weiter ausgewertet. Als Ergebnis werden potentiell interagierende Dreieckspaare zurückgegeben.

Schnitt- und Berührtests mit Indexierung

In den räumlichen Operatoren bewirkt die Identifikation interagierender Dreiecke die quadratische Laufzeitkomplexität. Um dem entgegen zu wirken, werden Funktionen für Schnitt- und Berührtests zwischen Dreiecken mit R*-Tree-Unterstützung eingeführt.

Der Funktion `IntersectionTestIndexed` werden die R*-Trees zweier räumlicher Entitäten übergeben. In einer parallelen Tiefensuche werden jeweils zwei Knoten der Bäume auf Überlappung sowie auf Enthaltensein getestet. Interagieren die Knoten räumlich miteinander, werden neue Paare aus den referenzierten Kindknoten gebildet und auf einen *Stapel* (engl. *stack*) geschoben. Besitzt ein Knoten keine Kinder, wird der Ursprungsknoten wiederverwendet. In der nächsten Iteration der `While`-Schleife wird das zuletzt hinzugefügte Paar vom Stapel genommen und die Prozessierung beginnt von neuem. Besteht ein Paar aus zwei Blattknoten, wird es der Funktion `TriangleIntersection` übergeben. In dieser werden die indizierten Dreiecke der Knoten dereferenziert und auf Schnitt getestet. Wird ein Schnitt erkannt, kann die Baumtraversierung abgebrochen werden.

Auflistung 5.9 zeigt den Pseudocode des Algorithmus. Abbildung 5.7 illustriert das Vorgehen beispielhaft an zwei übersichtlichen Geometrie- und Baumstrukturen.

```
bool IntersectionTestIndexed(RTree A, RTree B)
  stack.Push(Pair(A.Root, B.Root))
  while (stack.Count > 0)
    pair = stack.Pop()
    if (pair.Item1.IsLeave && pair.Item2.IsLeave)
      if TriangleIntersection(pair)
        return true
    foreach (var childA in pair.Item1.MeOrMyChilden)
      foreach (var childB in pair.Item2.MeOrMyChilden)
          if(childA.Bounds.Intersect(childB.Bounds)
            stack.Push(Pair(childA, childB))
return false
```

Auflistung 5.9: Dreieck-Schnitttest zwischen zwei BReps mit Indexierung

Im entsprechenden Berührtest wird die Funktion `TriangleIntersection` durch `TriangleMeeting` ersetzt. Ansonsten sind beide Algorithmen identisch aufgebaut.

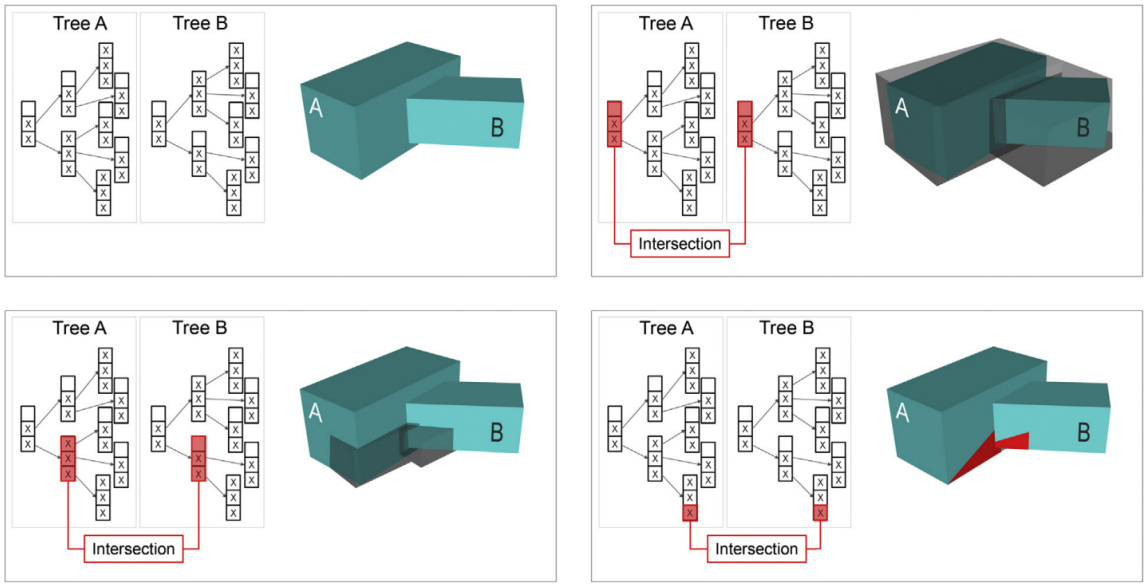


Abbildung 5.7: Demonstration einer parallelen Baumtraversierung zweier indexierter BRep-Strukturen.

Inside/Outside-Klassifizierung mit Indexierung

Die Inside/Outside-Klassifizierung von Dreiecken und übergeordneten BRep-Strukturen basiert auf der Aussendung von Teststrahlen und Zählung der auftretenden Dreiecksschnitte. Zur Bestimmung der Schnittanzahl müssen im ursprünglichen `IsInside`-Algorithmus alle Dreiecke der potentiell umhüllenden Entität mit jedem Strahl verschnitten werden. Dies hat eine lineare Zeitkomplexität bezüglich der mittleren Dreiecksanzahl n zur Folge. Die bereits in den Schnitt- und Berührtests genutzte R*-Tree-Indizierung kann auch in diesem Fall eingesetzt werden.

Die Integration der Indexstruktur in die Inside/Outside-Klassifizierung basiert auf der Vorselektion von Dreieckskandidaten durch eine angepasste Bounding Box. Abbildung 5.8 illustriert das Vorgehen beispielhaft anhand der zwei BRep-Strukturen A und B.

Aus A wird ein Dreieck ausgewählt (b). Durch Untersuchung der Normale des Dreiecks wird die Richtung mit der geringsten Abweichung parallel zum Koordinatensystem ermittelt und zur Strahlaussendung genutzt. Die Bounding Box des Dreiecks wird auf den Maximal- bzw. Minimalwert der Bounding Box von Entität B in die entsprechende Richtung vergrößert (c). Diese erweiterte Box wird anschließend als Suchkriterium für eine Abfrage des R*-Trees von B genutzt (d & e). Abschließend müssen die ermittelten Dreieckskandidaten auf Schnitt mit dem Strahl geprüft werden (f). Auflistung 5.10 zeigt den Algorithmus `IsInsideIndexed` als Pseudocode.

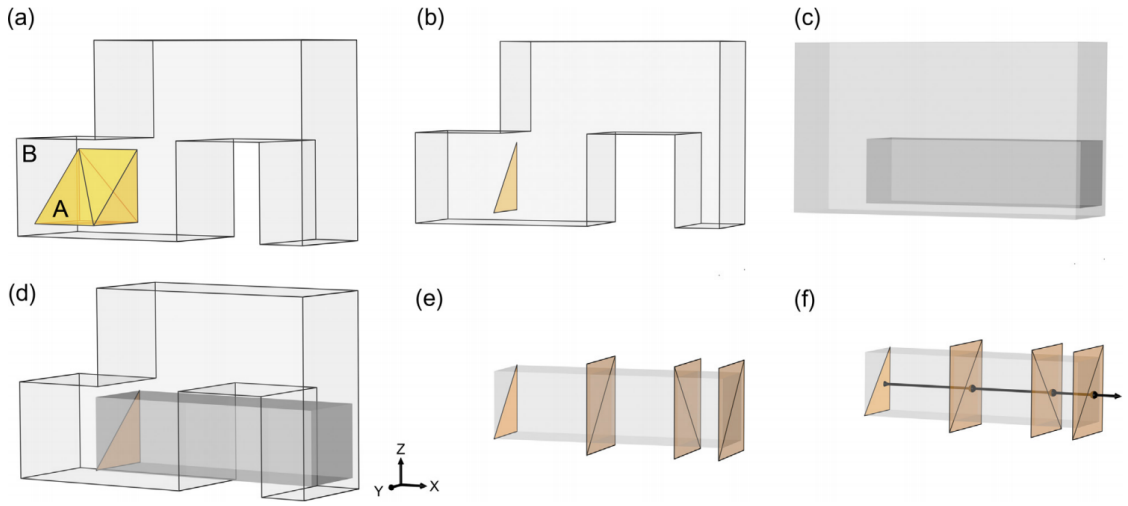


Abbildung 5.8: R*-Tree-Integration in die strahlbasierte Inside/Outside-Klassifizierung

```
bool IsInsideIndexed(Triangle TriangleA, RTree TreeB)
  direction = NearestParallelDirection(TriangleA.Normal)
  extendedTriBoxA = ExtendBox(TriangleA.Bounds, TreeB.Root.Bounds, direction)
  triangleCandidates = TreeB.Overlap(extendedTriBoxA)
  return IsInside(TriangleA, triangleCandidates)
```

Auflistung 5.10: Strahlbasierte Inside/Outside-Klassifizierung mit räumlicher Indizierung

Direktionale Operatoren mit Indexierung

Wie erläutert basieren die direktionalen Operatoren auf der Auswertung direktionaler Dreiecksbeziehungen. Um die Zeitkomplexität dieser Untersuchung zu verringern, wird ein Verfahren ähnlich der R*-Tree-unterstützten Inside/Outside-Klassifikation genutzt.

In der relaxierten Variante werden Prismen-Dreieck-Schnitte untersucht. Anstatt jedes Prisma des Referenzobjekts mit jedem Dreieck des Testobjekts zu untersuchen, wird die Bounding Box des Prismas für eine Kandidatenauswahl genutzt. Der R*-Tree von B gibt somit die in Frage kommenden Dreiecke zurück, die auf Schnitt mit dem Prisma untersucht werden müssen.

Eine Vorauswahl von Dreiecken durch eine erweiterte Bounding Box und einen R*-Tree kann auch in der strikten Variante der direktionalen Operatoren genutzt werden. Das betrifft sowohl die analytische, als auch die heuristische Implementierung. Im ersten Fall werden aus allen Dreieckskandidaten zusammenhängende Polygone gebildet und ein Dreieck-In-Polygon-Test ausgeführt. Im heuristischen, strahlbasierten Ansatz werden die vom Testdreieck ausgesendeten Strahlen gegenüber der reduzierten Menge an Dreieckskandidaten getestet.

Auflistung 5.11 zeigt den Algorithmus `AboveRelaxedIndexed` als Pseudocode. Auflistung 5.12 zeigt den Algorithmus des strikten `Above`-Operators in der analytischen Umsetzung. Die heuristische, strikte Variante ist in Auflistung 5.12 aufgeführt.

```
boolean AboveRelaxedIndexed(BRep A, BRep B)
  if Overlaps(A,B)
    return true
  maxZ = B.Bounds.MaxZ
  foreach triA in A
    prismA = new Prisma(triA, maxZ, Direction.ZPositiv)
    triCandidates = B.Tree.Overlap(prismA.Bounds)
    foreach triB in triCandidates
      if Intersects(prismA, triB)
        return true
  return false
```

Auflistung 5.11: BRep-basierter Algorithmus zur Bestimmung relaxierter, direktionaler Prädikate mit räumlicher Indizierung (Above-Variante)

```
boolean AboveStrictAnalyticIndexed(BRep A, BRep B)
  if Pretest(A,B) //overlaps test and extended bounding box test
    return false
  foreach triangleB in B
    extendedBoxB = triangleB.Bound.Extend(A.Bound, Direction.YNegative)
    proPolysA = List
    proPolysA.Add(A.RTree.Intersect(extendedBoxB))
    if(!proPolysA.InsideContour(triangleB.ProjectOnXY))
      return false
  return true
```

Auflistung 5.12: Analytischer BRep-basierter Algorithmus zur Bestimmung strikter, direktionaler Prädikate mit räumlicher Indizierung (Above-Variante)

```
boolean AboveStrictHeuristic(BRep A, BRep B)
  if Pretest(A,B) //overlaps test and extended bounding box test
    return false
  foreach triangleB in B
    triangleB.CreateSamplingPoints()
    extendedBoxB = triangleB.Bound.Extend(A.Bound, Direction.YNegative)
    triangleCandidatesA = RTree.Intersect(extendedBoxB))
    foreach point in triangleB.SamplingPoints
      ray = Ray(point, Direction.YNegative)
      if(!triangleCandidatesA.IntersectAnyTriangle(ray))
```

```
        return false
    return true
```

Auflistung 5.13: Heuristischer BRep-basierter Algorithmus zur Bestimmung strikter, direktionaler Prädikate mit räumlicher Indizierung (Above-Variante)

Metrische Operatoren mit Indexierung

Die indexierte Implementierung der metrischen Operatoren `Nearer` und `Farther` nutzt einen R*-Tree-basierten *Nearest-Neigbour-Search*-Algorithmus. Dieser orientiert sich an der *Punkt-Feature-Methode* aus (Roussopoulos et al., 1995) und weist Ähnlichkeiten mit der parallelen Baumtraversierung des Schnitt/Berühr-Tests auf. Hier müssen anstelle des einfachen Überlappungstests zwischen Boxen minimale Distanzen und *minimale Maximaldistanzen* ausgewertet werden.

Die Berechnung der minimalen Maximaldistanz basiert dabei auf folgendem Grundsatz: Jedes Subelement einer n-dimensionalen Bounding Box enthält mindestens einen Punkt der indizierten Geometrie. Im zweidimensionalen Fall ist das Subelement eine Kante, in 3D ein Rechteck und in höheren Dimensionen eine *Hyperfläche*. Für jede Raumrichtung wird das zum Suchpunkt nähere Subelement ausgewählt. Anschließend wird für jedes Subelement die Maximaldistanz zwischen Suchpunkt und dem Subelement berechnet. Die minimale Maximaldistanz aus diesen Berechnungen dient schließlich als obere Grenze in der Reduktion der Kandidatenpaare. Abbildung 5.9 zeigt minimale Distanzen und minimale Maximaldistanzen zwischen einem Suchpunkt und mehreren Boxen.

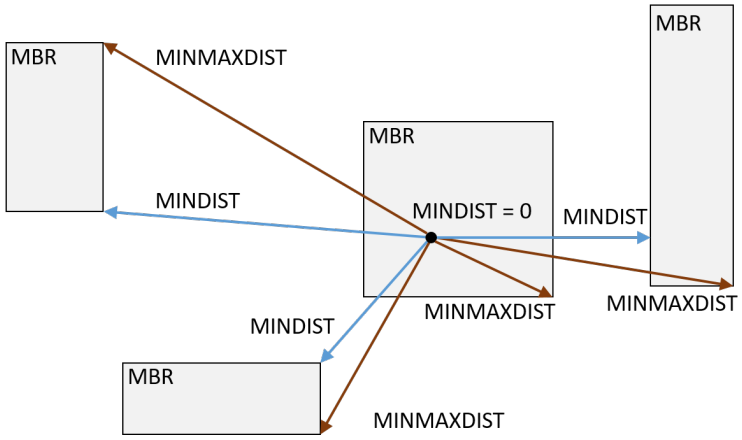


Abbildung 5.9: Minimale Distanzen und minimale Maximaldistanzen zwischen einem Suchpunkt und Bounding Boxen, nach Roussopoulos et al. (1995).

Der Algorithmus lässt sich auf die Distanzauswertung zwischen mehrdimensionalen Entitäten ausweiten. Dazu werden nicht die Distanzen zwischen Suchpunkt und Bounding Box untersucht, sondern die Distanzen zwischen zwei Boxen betrachtet.

Die *Reduktion von Kandidatenpaaren* (engl. *pruning*) erfolgt in einer Tiefensuche. Roussopoulos et al. (1995, Kap. 3.1) identifizieren drei Möglichkeiten zum Pruning. Diese sind im Folgenden verkürzt wiedergegeben, wobei MBR für *Minimal Bounding Box* steht und P den Suchpunkt bezeichnet. *Downward* bzw. *upward pruning* beschreibt, ob die Reduktion während absteigender oder aufsteigender Baumtraversierung angewendet wird.

1. An MBR M with MINDIST(P,M) greater than the MINMAXDIST(P,M') of another MBR M' is discarded (downward pruning).
2. An actual distance from P to a given object O which is greater than the MINMAXDIST(P,M) for an MBR M can be discarded (downward pruning).
3. Every MBR M with MINDIST(P,M) greater than the actual distance from P to a given object O is discarded (upward pruning).

Für die metrischen Operatoren mit einer oberen bzw. unteren Grenze ergeben sich zusätzliche globale Abbruchkriterien. Diese können genutzt werden, wenn sich das Intervall aus minimaler Distanz und minimaler Maximaldistanz komplett unterhalb bzw. oberhalb der Grenze befindet. Im Fall von `Nearer` kann die Untersuchung somit sofort mit `true` beendet werden, wenn die minimale Maximaldistanz unterhalb des Wertes des `bound`-Parameters liegt.

Auflistung 5.14 zeigt den Pseudocode für den `Nearer`-Operator mit R*-Tree-Integration.

```
boolean Nearer(BRep A, BRep B, double upperBound)
  stack = Stack();
  stack.Push(Pair(A.RTree,B.RTree));
  while(stack.Count > 0)
    pair = stack.Pop()
    itemA = pair.Item1
    itemB = pair.Item2
    if (!itemA.CanSubdivide && !itemB.CanSubdivide)
      minDist = GetDistance(itemA.Item, itemB.Item) //triangle distance
      if(minDist > currentMinMaxDist)
        continue //downward pruning 2
      if(minDist < currentMinDistTri)
        currentMinDistTri = minDist
    else
      foreach childA in MeOrMyChildren(itemA)
        foreach childB in MeOrMyChildren(itemB))
          minDist, minMaxDist = GetDistances(childA, childB)
          if(minDist > currentMinMaxDist)
            continue //downward pruning 1
          if(minDist > currentMinDistTri)
            continue //upward pruning
          if(minDist < currentMinDist)
```

```
            currentMinDist = minDist
          if(minMaxDist < currentMinMaxDist)
            currentMinMaxDist = minMaxDist
          stack.Push(childA, childB));
    if(currentMinMaxDist < upperBound || currentMinDistTri < upperBound)
      return true //global termination criterion
  return false
```

Auflistung 5.14: Nearer-Operator mit R*-Tree-basierter Nearest-Neigbour-Search

In diesem Abschnitt wurde die Integration der R*-Tree-basierten Indexierung in die topologischen, direktionalen und metrischen Operatoren beschrieben. Dadurch können komplexe Geometrierepräsentationen effizient verarbeitet werden. Im Zuge der Validierung der entwickelten BIM-Anfragesprache in Kapitel 8.2 wird dies durch empirische Laufzeitanalysen belegt.

5.2.4 Toleranzunterstützende Operatoren

Durch die Kombination von BRep-Prozessierung und R*-Tree-basierter Indizierung kann eine effiziente Auswertung räumlicher Relationen erreicht werden. Die direktionalen und topologischen Operatoren weisen jedoch eine eingeschränkte Anwendbarkeit auf, da geringste geometrische Abweichungen zwischen Entitäten zu fehlerhaften Ergebnissen führen können.

Auflistung 5.15 zeigt einen Ausschnitt aus der TranslaTUM-IFC-Datei, der minimal abweichende Vertices unterschiedlicher Entitäten enthält. Diese geringfügigen Differenzen sind durch die verschachtelten Koordinatensysteme von Gebäudemodellen zu erklären, die zu mehrfachen Koordinatentransformationen führen.

```
#...= IFCCARTESIANPOINT((0.,87.4999999999996));
#...= IFCCARTESIANPOINT((0.,87.5000000000001));
#...= IFCCARTESIANPOINT((0.,87.5000000000004));
```

Auflistung 5.15: Minimale Abweichungen zwischen Punktkoordinaten in der TranslaTUM-IFC-Datei

Daher werden toleranzunterstützende, topologische und direktionale Operatoren vorgeschlagen. Durch diese wird die Robustheit des Ansatzes bei numerischen Abweichungen gesteigert. Außerdem kann der Endnutzer somit semantische Toleranzen je nach Anwendungsfall nutzen. Um diese Operatoren umzusetzen, wird eine spezielle BRep-Struktur vorgeschlagen. Sie basiert auf der Erweiterung des ursprünglichen Dreiecknetzes durch die Erzeugung einer nach innen bzw. außen versetzten Begrenzung. Die Distanz zwischen innerer und äußerer Begrenzung wird über den Operatorparameter `tol` angegeben.

Abbildung 5.10 verdeutlicht das Vorgehen anhand einer zweidimensionalen Region A. Die innere Begrenzung erhält als Bezeichnung den Index i (engl. inner), die äußere den Index o (engl. outer).

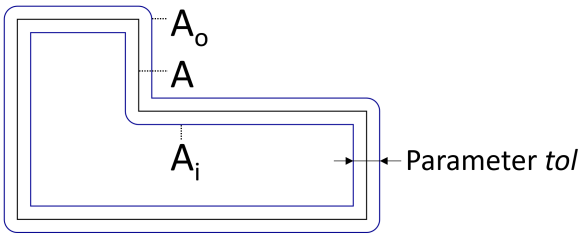


Abbildung 5.10: Erweiterung des ursprünglichen Dreiecknetzes durch Erzeugung nach innen und außen versetzter Begrenzungen (zweidimensionale Darstellung)

Derartige versetzte BRep-Strukturen können auf unterschiedliche Weise erzeugt werden (Rossignac & Requicha, 1986; Chen et al., 2005; Malosio et al., 2009). Für die Umsetzung der Operatoren wird in dieser Arbeit die *Multiple Normal Vectors of a Vertex*-Methode genutzt (Kim et al., 2004). Sie stellt einen Kompromiss hinsichtlich Robustheit der Geometrieerstellung und des Rechenaufwands dar. Die Methode nutzt die Normale eines Dreiecks als Richtung zu dessen Versetzung. Nach der Versetzung erfolgt eine Betrachtung der entstehenden Zwischenräume und Dreiecksüberschneidungen. Dabei wird zwischen weichen und scharfen Übergängen zwischen Kanten und Vertices unterschieden.

An **weichen Übergängen** mit ähnlichen Normalen werden die Zwischenräume und Dreiecksüberschneidungen durch *Verschmelzungen* beseitigt. An **scharfen Übergängen** werden Zwischenräume durch zusätzliche Übergangsdreiecke geschlossen. Überschneidungen werden in diesem Fall durch *Trimmen* von Dreiecken entfernt. Abbildung 5.11 zeigt die mit diesem Verfahren erstellte äußere Begrenzung für einen dreidimensionalen Körper.

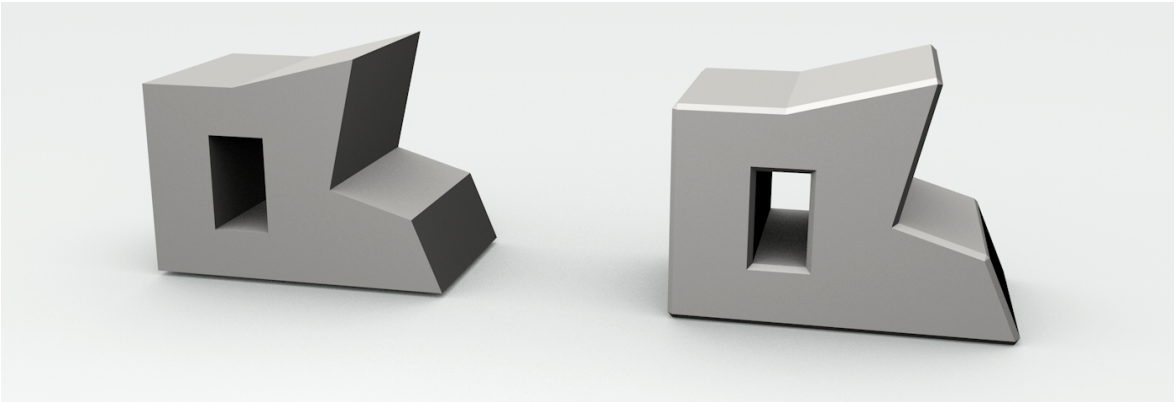


Abbildung 5.11: Äußere Begrenzung eines dreidimensionalen Körpers, erstellt mit der Multiple Normal Vectors of a Vertex-Methode

Zu deformierten Dreiecksnetzen kann es in diesem Verfahren kommen, wenn Selbstüberschneidungen auftreten und Dreiecke in der Größenordnung der Toleranz in der Ursprungsgeome-

trie enthalten sind. Im betrachteten Anwendungsfall ist der zu unterstützende Toleranzbereich jedoch im Verhältnis zur Ursprungsgeometrie in der Regel gering. Dies reduziert die Wahrscheinlichkeit, dass Selbstüberschneidungen auftreten. Zudem können Dreiecke in der Größenordnung des Toleranzbereichs durch eine Vorprozessierung entfernt werden. Wird ein erweiterter Bereich für Toleranzen benötigt, müssen andere, rechenintensivere Versetzungsverfahren verwendet werden (Chen et al., 2005).

Die toleranzunterstützenden, topologischen Operatoren benötigen wie ihre ursprünglichen Varianten eine formale Definition. Würde der Bereich zwischen und inklusive der inneren und äußeren Begrenzung als Randregion definiert werden, ergeben sich mitunter Abweichungen zum 4-IM. Abbildung 5.12 illustriert die Problematik anhand der Egenhofer'schen Definitionen für `touches` und `overlaps`. Die in (a) dargestellte räumliche Konstellation zeigt einen Schnitt zwischen Innerem und erweitertem Rand der Entitäten, jedoch kein Schnitt zwischen beiden Inneren. Dadurch sind weder `touches` noch `overlaps` erfüllt (b).

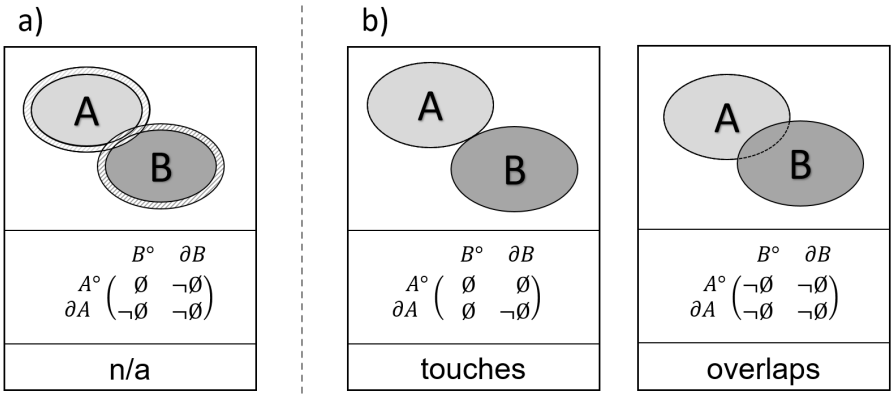


Abbildung 5.12: Zum 4-IM abweichende Matrix bei der Verwendung zusammengesetzter Randregionen

Die sich ergebenden Relationen zwischen zwei räumlichen Entitäten mit jeweils innerer und äußerer Begrenzung wurden von Clementini & Di Felice (1997) ausführlich untersucht. Dabei wurden 44 mögliche Relationen identifiziert. Die Anzahl der topologischen Prädikate wird anschließend durch ein Clustering auf 18 reduziert. Dennoch werden die sich ergebenden Prädikate im Kontext einer räumlichen BIM-Analyse als zu differenziert angesehen. Anstelle der Einführung neuer Vorlagen-Matrizen wird daher ein dreistufiges Verfahren zur Definition der toleranzunterstützenden, topologischen Operatoren entwickelt. Dieses nutzt ausschließlich die bereits eingeführten acht Egenhofer'schen 4-IM-Matrizen und die geometrischen Operationen zur Erstellung einer nach innen bzw. außen verschobenen Begrenzung.

Das Verfahren beginnt mit den ursprünglichen Geometrierepräsentationen und prüft, ob eines von mehreren topologischen Prädikaten erfüllt ist. Ist keines der erlaubten Prädikate zutreffend, wird die Untersuchung abgebrochen. Ist die erste Bedingung hingegen erfüllt, wird eine modifizierte Begrenzung erstellt. Ob nach innen und/oder außen verschoben wird, ist vom je-

weiligen Prädikat und vom Operanden abhängig. Im letzten Schritt muss mit der veränderten Geometrie wiederum eines von mehreren Prädikaten bestätigt werden. Abbildung 5.13 zeigt für jeden Operator die zu Beginn zulässigen Prädikate (1), die anzuwendende Geometrieerzeugung pro Operand (2) und die finalen Prädikate (3).

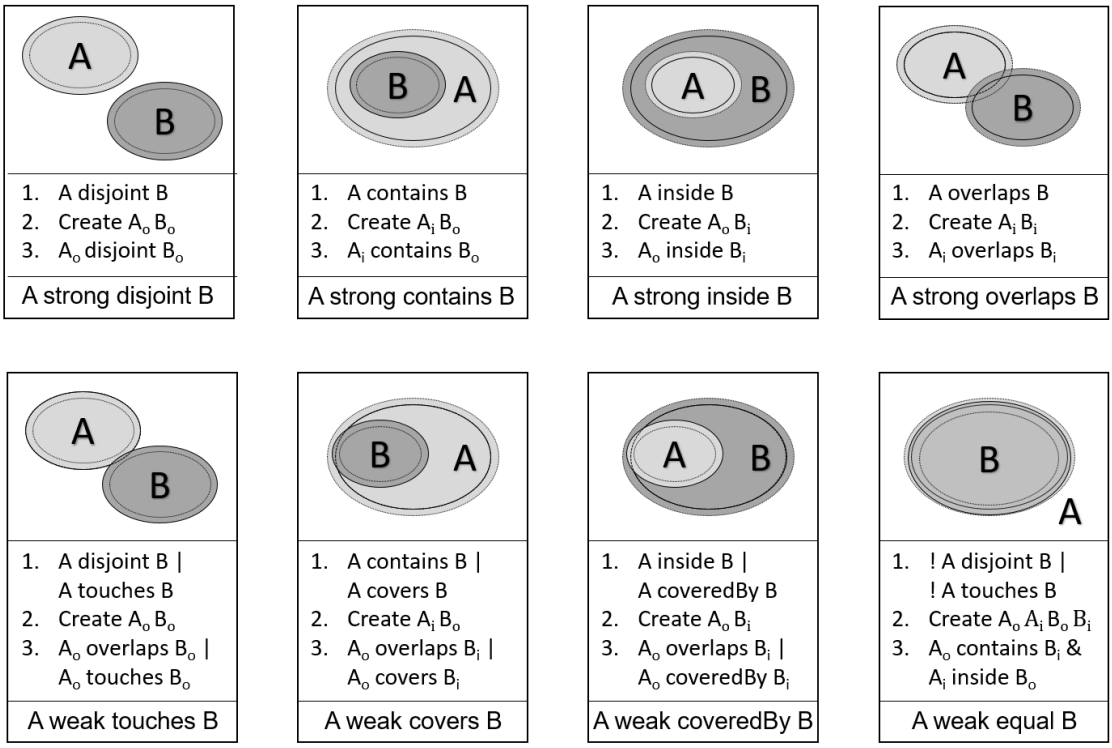


Abbildung 5.13: Toleranzunterstützende, topologische Operatoren durch zweifache topologische Prüfung und zwischengelagerter Geometriemodifikation

Bei der Nutzung dieser dreistufigen Definition ergeben sich zwei Gruppen an Operatoren. In der ersten tritt das topologische Prädikat **trotz** der Einführung der Toleranzen auf. In der zweiten Gruppe bestätigt sich das Prädikat **wegen** der angewendeten Toleranzen. Die erste Gruppe umfasst die Prädikate `disjoint`, `contains`, `inside` und `overlaps`, die Zweite `touches`, `covers`, `coveredBy` und `equal`.

Diese Operatoren ermöglichen das gezielte Auffinden bestimmter, topologischer Gegebenheiten, welche für eine Vielzahl räumlicher Fragestellung in der BIM-basierten Planung relevant sind. Über `strong disjoint` lassen sich topologisch stark getrennte Entitätspaare identifizieren. Mit `weak touches` können sich berührende Paare gefunden werden und zwar auch, wenn minimale Zwischenräume oder Überlagerungen vorliegen. Im Kontrast dazu kann mit `strong overlaps` eine starke topologische Überlagerungen ausgemacht werden. In vielen Fällen wird dieses topologische Prädikat auf einen Planungsfehler hinweisen, beispielsweise wenn ein Durchbruch nicht moduliert wurde und sich Bauteile durchdringen. `strong inside` und `strong overlaps` bzw. `weak covers` und `weak coveredBy` dienen der Selektion von ineinan-

der beherbergten Entitäten, beispielsweise bei der Verschachtelung räumlicher Strukturen wie Stockwerken, Räumen und Zonen. Über die schwachen Operatoren `covers` bzw. `coveredBy` lassen sich dabei geringe Überscheidungen und Zwischenräume bei Berührungen von Innen kompensieren. Die starken Operatoren `inside` bzw. `overlaps` selektieren hingegen verschachtelte Entitätspaare ohne gegenseitige Rand-Rand-Interaktionen. Der `weak equal` Operator ermöglicht die Selektion geometrisch identischer aber auch minimal veränderter Entitäten.

Auf Basis der dreistufigen Definition wird im Folgenden eine mögliche Implementierung der toleranzunterstützenden, topologischen Operatoren in Kombination mit der R*-Tree-Indizierung beschrieben. Dazu wird die Funktion `GetOffsetTriangles` eingeführt und die Funktion `IsInsideIndexed` erweitert. Die Signaturen der Funktionen sind in Auflistung 5.16 dargestellt.

```
list GetOffsetTriangles(Triangle Triangle, double tol, boolean outer)
bool IsInsideIndexed(Triangle TriangleA, BRep BRepB, double tol)
```

Auflistung 5.16: Hilfsfunktionen für die Umsetzung der toleranzunterstützenden, topologischen Operatoren

In den toleranzunterstützenden Operatoren müssen nicht zwangsläufig die kompletten inneren und äußeren Begrenzungen verarbeitet werden. Daher werden diese nur für lokale Bereiche erstellt, in denen potentielle Interaktionen durch die R*-Tree-Prozessierung erkennbar sind. Die Funktion `GetOffsetTriangles` liefert dazu für jedes Ursprungsdreieck eine Liste an Dreiecken. In dieser sind das versetzte Dreieck und berührende Verblendungsdreiecke enthalten. Der Wert des Versatzes wird über den Parameter `tol` gesteuert. Ob nach innen oder nach außen versetzt wird, bestimmt der Parameter `outer`.

Um eine zusätzliche R*-Tree-Erstellung für die versetzte Geometrie zu vermeiden wird der Baum der Ursprungsgeometrie wiederverwendet. Wird nach außen versetzt, müssen bei Schnitttests die Knoten entsprechend der verwendeten Toleranz vergrößert werden. Auch die Funktion `IsInsideIndexed` muss für die Toleranzunterstützung angepasst werden. So muss der Bereich für die Dreieckssuche im R*-Tree des potentiell umhüllenden BReps um den Toleranzwert vergrößert werden. Die gefundenen Dreiecke müssen vor dem Strahltest durch die versetzten Dreiecke ersetzt und Verblendungsdreiecke in die Untersuchung mit einbezogen werden. Auflistung 5.17 zeigt den Algorithmus für die Umsetzung des `contains`-Operators, Auflistung 5.18 den Algorithmus für den `touches`-Operator.

```
boolean StrongContains(BRep A, BRep B, double tol) //A strongly contains B
  if !Contains(A, B)
    return false
  stack.Push(Pair(A.Root, B.Root))
  while (stack.Count > 0)
    pair = stack.Pop()
```

```
    if (pair.Item1.IsLeave && pair.Item2.IsLeave)
      trisAi = GetOffsetTriangles(pair.Item1, tol, false)
      trisBo = GetOffsetTriangles(pair.Item2, tol, true)
      foreach triAi in trisAi
        foreach triBo in trisBo
          if TriangleIntersection(triAi, triBo)
            return false
      foreach (var childA in pair.Item1.MeOrMyChilden)
        foreach (var childB in pair.Item2.MeOrMyChilden)
            if(childA.Bounds.Intersect(childB.Bounds.Offset(tol))
              stack.Push(Pair(childA, childB))
  offsetTriB = trisBo = GetOffsetTriangles(B.Triangles[0], tol, true)
  return IsInside(offsetTriB, A)
```

Auflistung 5.17: Algorithmus des toleranzunterstützenden Contains-Operators

```
boolean WeakTouches(BRep A, BRep B, double tol) //A weakly touches B
  if Touches(A, B)
    return true
  if !Disjount(A, B)
    return false
  stack.Push(Pair(A.Root, B.Root))
  while (stack.Count > 0)
    pair = stack.Pop()
    if (pair.Item1.IsLeave && pair.Item2.IsLeave)
      trisAo = GetOffsetTriangles(pair.Item1, tol, true)
      trisBo = GetOffsetTriangles(pair.Item2, tol, true)
      foreach triAo in trisAo
        foreach triBo in trisBo
          if TriangleIntersection(triAo, triBo)
            return true
      foreach (var childA in pair.Item1.MeOrMyChilden)
        foreach (var childB in pair.Item2.MeOrMyChilden)
            if(childA.Bounds.Offset(tol).Intersect(childB.Bounds.Offset(tol))
              stack.Push(Pair(childA, childB))
  return false
```

Auflistung 5.18: Algorithmus des toleranzunterstützenden Touches-Operators

Die toleranzunterstützenden, direktionalen Operatoren werden ebenfalls auf Basis einer modifizierten BRep-Struktur definiert. Hier wird stets eine nach innen versetzte Begrenzung des Referenzobjekts A erstellt. Das Testobjekt B bleibt unverändert. Der Wert des Versatzes wird über den Parameter *tol* gesteuert.

Auflistung 5.19 zeigt den Algorithmus für die Umsetzung des `AboveRelaxed`-Operators. Die heuristische Variante für den toleranzunterstützenden, strikten Operator ist in Auflistung 5.20 dargestellt.

```
boolean WeakAboveRelaxed(BRep A, BRep B, double tol)
  if Overlaps(A,B, tol)
    return true
  maxZ = B.Bounds.MaxZ
  foreach triA in A
    triAis = GetOffsetTriangles(triA, tol, false)
    foreach triAi in triAis
      prismA = new Prisma(triAi, maxZ, Direction.ZPositiv)
      triCandidates = B.Tree.Overlap(prismA.Bounds)
      foreach triB in triCandidates
        if Intersects(prismA, triB)
          return true
  return false
```

Auflistung 5.19: Toleranzunterstützender, BRep-basierter Algorithmus zur Bestimmung relaxierter, direktionaler Prädikate (Above-Variante)

```
boolean WeakAboveStrictHeuristic(BRep A, BRep B, double tol)
  if Pretest(A,B, tol) //weak overlaps test and extended bounding box test
    return false
  foreach triB in B
      triB.CreateSamplingPoints()
      extendedBoxB = triangleB.Bound.Extend(A.Bound, Direction.YNegative)
      triangleCandidatesA = RTree.Intersect(extendedBoxB.Offset(tol)))
      foreach triA in triangleCandidatesA
        triAis = GetOffsetTriangles(triA, tol, false)
            foreach point in triB.SamplingPoints
              ray = Ray(point, Direction.YNegative)
              if(!triAs.IntersectAnyTriangle(ray))
                return false
  return true
```

Auflistung 5.20: Toleranzunterstützender, heuristischer BRep-basierter Algorithmus zur Bestimmung strikter, direktionaler Prädikate (Above-Variante)

Die Berechnung topologischer Prädikate aus BRep-Geometrie bedarf genauerer Betrachtung. So bewirkt eine minimale, numerische Änderungen von Koordinaten, dass Berührungen in Überschneidungen oder zur Disjunktheit übergehen. Dadurch schwindet die Aussagekraft topologischer Prädikate in ingenieurtechnischen Fragestellungen. Die hier vorgestellten Algorithmen unterstützen hingegen benutzerdefinierte Toleranzen, wodurch Operatoren je nach Fragestellung durch den Endnutzer angepasst werden können. Somit können topologische Prädikate in technischen Problemstellungen als robuste Auswahlkriterien für BIM-Daten eingesetzt werden.

5.3 Operatoren zur Datenextraktion und -integration

Bauprojekte bedürfen der intensiven Kollaboration von Fachplanern. Zur Übergabe von Projektdaten werden in einer BIM-basierten Arbeitsumgebung derzeit noch Dateien verwendet (Metthews et al., 2011, S. 16). Idealerweise kommen dabei offene Standards wie die IFC zum Einsatz.

Projektdaten müssen vor der Weitergabe an die jeweiligen Fachplaner aufbereitet werden um bestimmten Vorgaben zu genügen. Benötigt wird somit auch Funktionalität um aus einem Gesamtmodell diejenigen Daten zu selektieren und in einem Teilmodell zu speichern, die für die anstehenden Arbeiten benötigt werden. In umgekehrter Weise müssen Teilmodelle unterschiedlicher Herkunft wieder zu einem Gesamtmodell vereinigt werden. In diesem Abschnitt werden die entsprechenden Operatoren für die Teilmodellerstellung und -reintegration präsentiert.

5.3.1 Teilmodellextraktion

Für die Erstellung von Teilmodellen werden zwei Operatoren vorgeschlagen, die auch in Kombination eingesetzt werden können. Der `ExportModel`-Operator dient zur schemakonformen Speicherung von Entitäten in einer neue Instanzdatei. Somit können direkt stark reduzierte Modelle gewonnen werden. Beispielsweise kann ein Modell aus allen tragenden Bauteilen des zweiten Stockwerks gebildet werden.

Für einen validen IFC-Export muss eine Grundstruktur in der erstellten Instanzdatei vorhanden sein. Diese setzt sich unter anderem aus einer Hierarchie räumlicher Strukturen zusammen. Daher werden im `ExportModel`-Operator die entsprechenden Strukturen für die Repräsentation von Gebäuden (IfcBuildings), Stockwerken (IfcBuildingStoreys) und Räumen (IfcSpaces) übernommen. Der Rest der Grundstruktur setzt setzt sich aus Angaben zu beteiligten Personen und Organisationen und weiteren Projekteigenschaften zusammen. Im Exportfall wird eine entsprechende Vorlage mit aktuellen Werten vervollständigt und in die erstellte Instanzdatei eingefügt werden. Zur Illustration zeigt Auflistung 5.21 relevante Teile der TranslaTUM-IFC-Datei.

```
#1  = IFCORGANIZATION($,'Autodesk Revit 2015 (ENU)',$,$,$);
#5  = IFCAPPLICATION(#1,'2015','Autodesk Revit 2015 (ENU)','Revit'); ...
#36 = IFCPERSON($,'Daum','Simon',$,$,$,$,$);
#38 = IFCORGANIZATION($,'','',$,$);
#39 = IFCPERSONANDORGANIZATION(#36,#38,$);
#42 = IFCOWNERHISTORY(#39,#5,$,.NOCHANGE.,$,$,$,1430232945); ...
#93 = IFCPROJECT(...,#42,'Nummer',$,$,'Name','Status',(#85),#80);
#104= IFCPOSTALADDRESS($,$,$,$,('Adresse'),$,'','M\X2\00FC\X0\nchen','','D');
#108= IFCBUILDING(...,#42,'',$,$,#33,$,'',.ELEMENT.,$,$,#104); ...
#123= IFCBUILDINGSTOREY(...,#42,'UG3',$,$,#121,$,'UG3',.ELEMENT.,-12260.); $
```

Auflistung 5.21: IFC-Grundstruktur am Beispiel der TranslaTUM-Datei

Für wiederkehrende Übergabeszenarien können, wie in Kapitel 3.4.2 beschrieben, die zu übergebenden Inhalte als Modellsichten definiert werden. Liegen derartige Sichten im mvdXML-Format vor, können sie rechnergestützt verarbeitet werden. Somit ist eine automatische Ableitung des entsprechenden Teilmodells möglich. Dazu müssen die ConceptTemplates der mvdXML-Datei interpretiert werden. Diese beinhalten Regeln zur Attributbelegung (AttributeRule) und Typisierung von Entitäten (EntityRule). Derartige Regeln lassen sich zudem über TemplateRules parametrisieren. Zur automatisierten Prozessierung dieser Regeln müssen Typen und Attribute von Entiäten überprüft werden. Dies sind Grundfunktionalitäten der angestrebten Anfragesprache, wodurch eine MVD-basierte Filterung umsetzbar wird (vgl. Kap. 6.5).

Auflistung 5.22 zeigt eine beispielhafte mvdXML-Datei zum Nutzung von Materialebenen. Der `MvdFilter`-Operator wertet die in der Modellsicht referenzierten ConceptTemplates aus. Somit können Anforderungen an Instanzen eines IFC-Typs abgeleitet werden und diese gleichzeitig aus einem Gesamtmodell inklusive der geforderten Referenzen extrahiert werden. Im hier abgebildeten Beispiel muss eine IfcSlabStandardCase bestimmte materialbezogene Entitäten referenzieren. Für die gesamte mvdXML-Datei sei auf (Weise, 2015) verwiesen. Die Anwendung des `MvdFilter`-Operators in Anfragen wird in Kapitel 8.1.3 erörtert.

```
...<SubTemplates>
 <ConceptTemplate name="Material Layer Set Usage" ...>
  <Rules>
   <AttributeRule AttributeName="HasAssociations" Cardinality="One">
    <EntityRules><EntityRule EntityName="IfcRelAssociatesMaterial" ...> ...
<Views>
 <ModelView name="CoordinationView" applicableSchema="IFC4">
  <ConceptRoot name= "Slab" applicableRootEntity="IfcSlabStandardCase">
    <Concepts><Concept name="Material Layer Set Usage"> ...
```

Auflistung 5.22: Ausschnitt aus einer beispielhaften mvdxml-Datei zum Nutzung von Materialien an Stützen, stark vereinfacht, nach (Weise, 2015)

5.3.2 Modellreintegration

Im vorherigen Abschnitt wurden zwei Operatoren vorgestellt, die zur Teilmodellextraktion dienen. Im Zuge der Planungsfortführung wird das so gewonnene Teilmodell erweitert und muss nach der Bearbeitung wieder in den Gesamtdatenbestand des Projekts integriert werden.

Die in dieser Arbeit verwendete Terminologie für die Zusammenführung von Datensätzen stammt aus dem Bereich der *textuellen Vergleichswerkzeuge*. Derartige Anwendungen werden beispielsweise in *kollaborativen Quellcode-Managementsystemen* eingesetzt (Smith, 1988). Werden mit einem Vergleichswerkzeug mehrere Datensätze unter Berücksichtigung lokaler Änderungen zusammengeführt, spricht man von *merging*. In diesem Prozess werden textuelle Daten als ursprünglich, neu hinzugefügt oder als verändert klassifiziert. Dadurch sind alle *Abweichungen* (engl. *diffs*) zwischen den verwendeten Datenquellen ersichtlich.

Sind keine Konflikte vorhanden kann automatisch der Gesamtdatenbestand mit allen Bearbeitungen abgeleitet werden. Konflikte können auftreten, wenn gleichzeitig an identischen Datenbereichen unterschiedliche Änderungen durchgeführt wurden. Derartige Konflikte müssen manuell aufgelöst werden.

Ein spezielle Variante der Datenzusammenführung ist das *three-way merging* (Lindholm, 2004). In diesem werden drei Datensätzen D_0, D_1 und D_2 verarbeitet. D_0 stellt den *base common ancestor*, also den gemeinsamen Vorgänger von D_1 und D_2 dar. Das heißt, dass D_1 und D_2 ursprünglich als Duplikate von D_0 entstanden sind. Beide Datensätze wurden zwischenzeitlich individuellen Änderungen unterzogen. Durch das merging werden D_1 und D_2 unter Berücksichtigung des gemeinsamen Vorgängers D_0 zum Datensatz D_m vereinigt. Abbildung 5.14 illustriert das Verfahren des three-way mergings anhand von ausschnittsweise betrachteten IFC-Dateien.

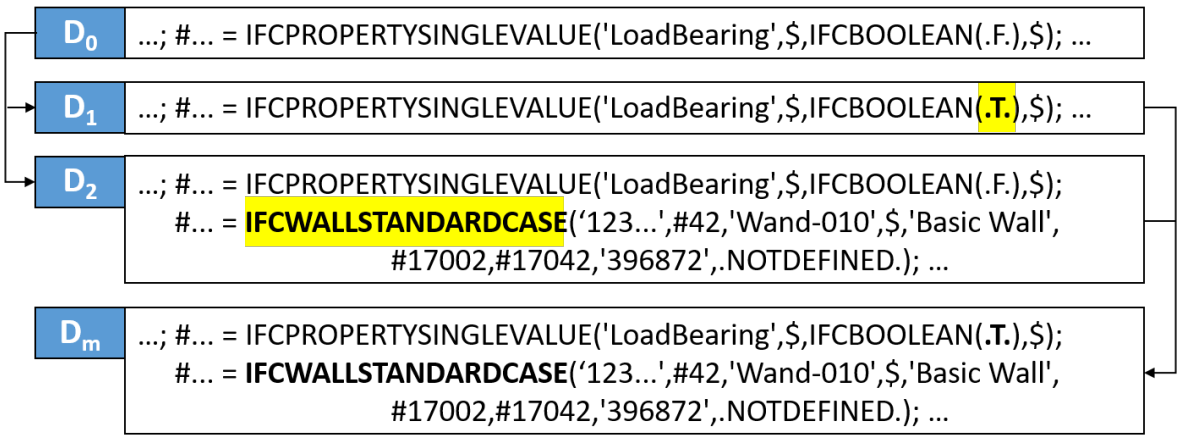


Abbildung 5.14: Three-way merging am Beispiel von IFC-Dateien

In D_1 wurde das *LoadBearing*-Attribut verändert und in D_2 eine zusätzliche Entität eingefügt. In D_m werden die Daten zusammengeführt. Hier zeigt sich der Vorteil des three-way gegenüber dem two-way merging. So kann die Änderung des *LoadBearing*-Attributs als Update erkannt werden und eine manuelle Konfliktauflösung durch den Endnutzer ist nicht nötig.

Unter Berücksichtigung der Datensätze D_0, D_1 und D_2 beschreibt Abbildung 5.15 den kollaborativen Planungsprozess in einer BIM-basierten Arbeitsumgebung. Der Arbeitsablauf beinhaltet die Teilmodellextraktion, die Modellerweiterung und die Zusammenführung der Teilmodelle zu einem Gesamtmodell.

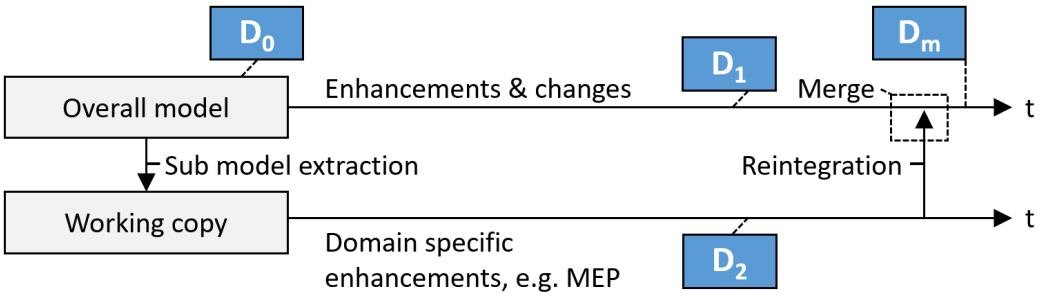


Abbildung 5.15: Teilmodellextraktion, Modellerweiterung und die Gesamtmodellerstellung

Werden D_1 und D_2 unter Berücksichtigung ihres gemeinsamen Vorgänger D_0 in einem three-way merging vereinigt, können folgende Fälle auftreten.

1. **Übernahme**: Das Datenelement ist in D_0, D_1 und D_2 unverändert vorhanden. → Das Datenelement wird in D_m übernommen.
2. **Löschung**: Ein Datenelement ist in D_0, aber nicht in D_1 oder D_2 vorhanden. → Das Datenelement wird nicht in D_m übernommen.
3. **Hinzufügung**: Ein Datenelement ist nicht in D_0, aber entweder in D_1 oder in D_2 vorhanden. → Das neue Datenelement wird in D_m übernommen.
4. **Update**: Ein Datenelement ist in D_0, aber verändert entweder in D_1 oder D_2 vorhanden. → Das veränderte Datenelement wird in D_m übernommen.
5. **Synchroner Update**: Ein Datenelement ist in D_0, aber verändert in D_1 und D_2 vorhanden. → Unterscheiden sich die Änderungen zwischen D_1 und D_2, besteht ein Konflikt, der manuell behoben werden muss.

In den fünf merging-Fällen ist die Identifikation zusammengehöriger Elemente nötig. In der IFC stellen Entitäten die betrachteten Datenelemente dar, wobei nur Subklassen von IfcEntity durch das *GlobalID*-Attribut eine eindeutige Identität besitzen. Wird davon ausgegangen dass alle GlobalIDs korrekt von D_0 in D_1 und D_2 übernommen wurden, kann ein bezeichnerbasiertes three-way merging durchgeführt werden.

Neben dieser rein attributiven Vereinigung besteht die Möglichkeit das Verfahren durch die Hinzunahme eines `Equals`-Vergleichs zu erweitern. Dadurch werden die Geometrierepräsentationen der Datenelement aus D_0, D_1 und D_2 auf topologische Gleichheit geprüft. Hierfür wird der toleranzunterstützende `Equals`-Operator aus Abschnitt 5.2.4 verwendet. Für die fünf merging-Fälle ergeben sich dadurch folgende Erweiterungen.

1. **Übernahme**: Durch `Equals` wird sichergestellt, dass sich die Geometrie der Entitäten nicht verändert hat.
2. **Löschung**: Wird das gesuchte Element nicht durch ID-Abgleich identifiziert, wird eine räumliche Analyse mit `Equals` ausgeführt. Somit kann eine Löschung von einer Veränderung unterschieden werden, auch wenn die *GlobalId*s der Entitäten nicht übereinstimmen.
3. **Hinzufügung**: Wird das gesucht Element nicht durch ID-Abgleich identifiziert, wird eine räumliche Analyse mit `Equals` ausgeführt. Somit kann eine Hinzufügung von einer Veränderung unterschieden werden, auch wenn die *GlobalId*s der Entitäten nicht übereinstimmen.
4. **Update**: Durch `Equals` wird erkannt, dass sich die Geometrie der Entitäten verändert hat. Die veränderte Geometrie kann übernommen werden.
5. **Synchroner Update**: Durch `Equals` wird erkannt, dass sich die Geometrie der Entitäten verändert hat. Zwei veränderte Geometrierepräsentationen werden als Konflikt erkannt.

Der beschriebene Mering-Operator vereint somit eine attributive mit einer toleranzunterstützenden, topologischen Betrachtung von BIM-Datensätzen zu deren Zusammenführung.

In diesem Abschnitt wurden domänenspezifische Operatoren zur Modellextraktion und Modellzusammenführung entwickelt. Mit dem MVDFilter-Operator lassen sich Teilmodelle auf Basis von mvdXML-Definitionen extrahieren. Zudem können Entitätsmengen, die durch andere Operatoren selektiert wurden, mit Hilfe des ExportModel-Operators zu validen IFC-Modellen exportiert werden. Der Merger-Operator realisiert ein neuartiges BIM-spezifisches three-way merging. Dieses untersucht nicht nur Attribute und wertet einen eindeutigen Entitätsbezeichner aus, sondern nutzt eine topologische Gleichheitsbetrachtung zwischen den räumlichen Repräsentationen der Teilmodelle.

5.4 Zusammenfassung

Gebäudemodelle sind Repräsentationen mit Raumbezug die mit zeitlichen Informationen erweitert werden können. Für die angestrebte, ganzheitliche Analyse derartiger Modelle ist es notwendig, zeitliche und räumliche Auswertungsfunktionalität in eine entsprechende BIM-Anfragesprache zu integrieren. In diesem Kapitel wurden dementsprechend Methoden und Algorithmen zur Realisierung metrischer, direktionaler und topologischer Operatoren entwickelt.

Um akzeptable Ausführungsgeschwindigkeiten bei der nötigen dreidimensionalen Analyse zu erreichen, wird eine BRep-Verarbeitung und eine räumliche Indizierung auf Basis von R*-Trees eingesetzt. So können effiziente räumliche Operatoren umgesetzt werden und die weiteren Nachteile oktalbaumbasierter Algorithmen umgangen werden. Die direktionalen und topologischen Operatoren wurden zudem erweitert, so dass benutzerdefinierte Toleranzen bei der Auswertung berücksichtigt werden können.

Die vorgestellten zeitlichen Operatoren ermöglichen die Analyse der baulichen Terminplanung. Dazu werden topologische Beziehungen zwischen Zeitintervallen ausgewertet, wobei grobe als auch feingranulare zeitliche Untersuchungen möglich sind. Neuartige Ansätze zur Modellextraktion und Modellreintegration komplettieren dieses Kapitel. Mit ihnen wird eine flexible Extraktion benötigter Daten und eine robuste Zusammenführung bearbeiteter Modelle möglich.

Kapitel 6

Konzeption einer Anfragesprache für Gebäudemodelle

Im vorherigen Kapitel wurden spezielle Operatoren entwickelt, die für die Analyse von 4D-Gebäudemodellen benötigt werden. Diese Operatoren stellen einen zentralen Teil der domänenspezifischen Query Language for 4D Building Information Models (QL4BIM) dar, die in diesem Kapitel vorgestellt wird. Als Einstieg werden Paradigmen textueller und visueller Sprachen erörtert und Anforderungen an eine Anfragesprache für BIM-Daten abgeleitet. Danach wird auf die Eigenschaften der entwickelten Sprache detailliert eingegangen. Diese ist direkt für den Einsatz durch Fachanwender ausgelegt und basiert auf einer reduzierten Anzahl an syntaktischen Regeln. Zudem bietet die Sprache neben einer textuellen auch eine visuelle Notation. Das Kapitel beinhaltet außerdem die Diskussion aller in der Sprache integrierten Operatoren und schließt mit der Implementierung der in Kapitel 4.2 definierten Musteranfragen.

6.1 Programmierparadigmen

Eine neuartige BIM-Anfragesprache kann nicht isoliert entwickelt werden. Stattdessen muss eine Orientierung am Stand der Forschung erfolgen. Daher werden im Anschluss allgemeine *Programmierparadigmen* und *Prinzipien visueller Sprachen* betrachtet. Auf diese wird bei der Einordnung von QL4BIM Bezug genommen.

6.1.1 Paradigmen textueller Sprachen

Anfragesprachen zur Informationsextraktion und zur strukturierten Datenmanipulation stellen Spezialisierungen allgemeiner Programmiersprachen dar. Zur Klassifizierung von Program-

miersprachen dienen sogenannte Programmierparadigmen. Diese gruppieren mathematische Grundlagen und formale Lösungsansätze, die je nach Sprache mehr oder weniger unterstützt werden. Abbildung 6.1 zeigt zentrale Programmierparadigmen und ihrer Zusammenhänge (Van Roy, 2007; Fischer & Hofer, 2011) .

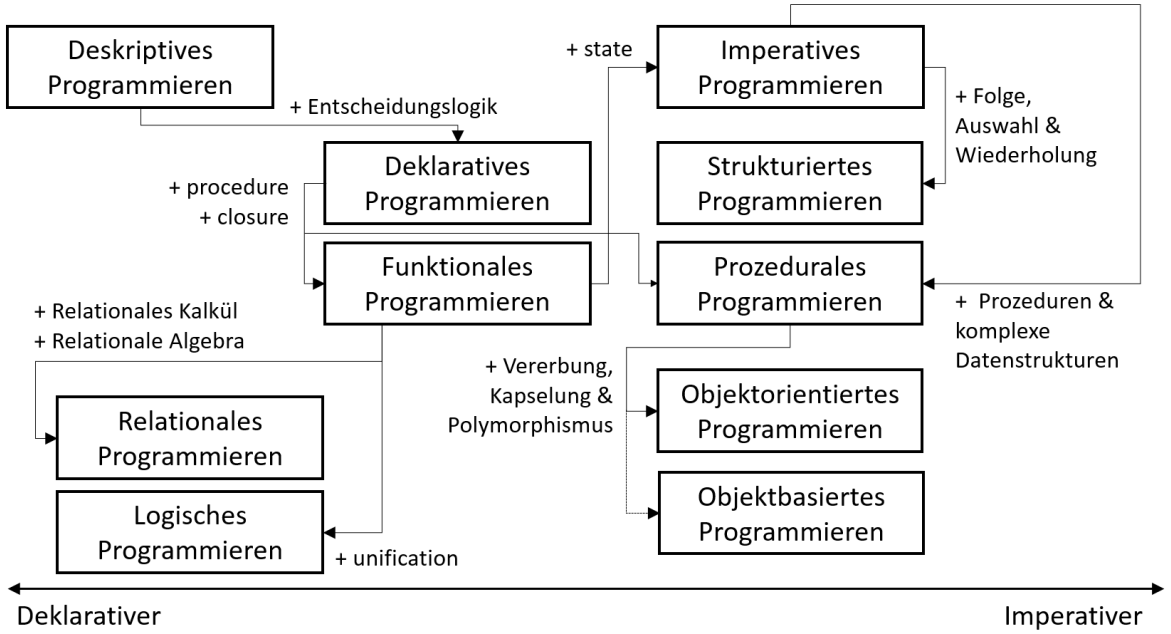


Abbildung 6.1: Programmierparadigmen nach (Van Roy, 2007; Fischer & Hofer, 2011, S. 700)

Deskriptives Programmieren: Dieses Paradigma wird teilweise mit einem *deklarativen* Ansatz gleichgesetzt, kann aber auch eine Programmierung ohne Entscheidungslogik bezeichnen. Dann sind Auszeichnungssprachen wie XML als Anwendungen *deskriptiver* Programmierung anzusehen (Fischer & Hofer, 2011).

Deklaratives Programmieren: In einer deklarativen Programmierung wird eine Aufgabe innerhalb einer passenden Theorie auf logischer Basis gelöst. Dies erhöht den Abstraktionsgrad der Umsetzung, wodurch das Resultat (*das Was*) und nicht die explizite Lösungsvorschrift (*das Wie*) spezifiziert werden muss. Unterkategorien sind das *logische*, das *funktionale* und das *relationale* Programmieren (Torgersson, 1996).

Funktionales Programmieren: Der Bezeichnung entsprechend repräsentieren Funktionen das zentrale Element dieser Programmiermethodik. Durch deren Verschachtelung können Daten von übergeordneten an untergeordnete Funktionen weitergeleitet werden. In einer strikten Version des Paradigmas können nur Funktionsparameter mit Werten belegt werden. Zusätzliche Variable existieren nicht und eine erneute Wertezuweisung an einen Parameter innerhalb der Funktion ist untersagt. Somit besitzen Funktionen ihre eigene *isolierte Umgebung* (engl. *closure*) und die Reihenfolge ihrer Ausführung ist

für das Ergebnis des funktionalen Programms irrelevant. Da die Neubelegung von Variablen nicht unterstützt wird, müssen übliche Kontrollstrukturen anderweitig realisiert werden. Beispielsweise können Schleifen in einer funktionalen Umgebung durch *rekursive Funktionen* umgesetzt werden (Michaelson, 2013, Seite 4-7).

Relationales Programmieren: Die Methodik weist Parallelen zum funktionalen Programmieren auf, basiert aber zusätzlich auf der *relationalen Algebra* bzw. dem *relationalen Kalkül*. Daten liegen somit in Relationen vor und können durch Operatoren bearbeitet werden. Des Weiteren können Programme selbst als Relationen abgebildet werden, wodurch auch diese durch Operatoren manipulierbar werden (MacLennan, 1983). Das relationale Paradigma hat jedoch in dieser Striktheit keine praktische Bedeutung erlangt. In abgeschwächter Form, in der Relationen die zentrale Datenstruktur darstellen, dient es als Basis für Anfragesprachen wie *QUEL* und SQL (Manthey, 2002). In diesen Anwendungsfällen wird der deklarative Ansatz der relationalen Programmierung ersichtlich. So wird der Programmablauf, wie beispielsweise die Iteration durch alle Tupel einer Relation nicht explizit angegeben.

Logisches Programmieren: Programme bestehen im Allgemeinen aus *Logik und Kontrollfluss* (Kowalski, 1979). Innerhalb dieses Paradigmas wird versucht durch Schlussfolgern auf die explizite Definition des Kontrollflusses verzichten zu können. Eine derartige *Deduktion* basiert in logischen Programmen auf der *Prädikatenlogik erster Stufe*. Diese nutzt ein Alphabet aus logischen und benutzerdefinierten Symbolen sowie Terme und Formeln aus diesem Alphabet. Derartige Elemente ermöglichen es, die logischen Eigenschaften des betrachteten Bereichs formal auszudrücken. Zur eigentlichen computerbasierten Deduktion werden Terme gleichgesetzt (engl. *unification*) und Terme und Variablen durch *Substitution* letztendlich an explizite Werte gebunden (Gabbrielli & Martini, 2010, Kapitel 12.1-12.4).

Imperatives Programmieren: Diese Programmierung basiert auf *Anweisungen* (engl. *statements*), die aus Befehlen aufgebaut sind. Der fundamentalste Befehl stellt die *Zuweisung* dar (engl. *assignment*). Durch sie werden Variablen an konkrete Werte gebunden, was sich wiederum auf den *Zustand* (engl. *state*) des Programms auswirkt. Eine Anweisung kann daher nicht nur wegen expliziter Ergebnisse aufgerufen werden. Stattdessen werden auftretende *Nebeneffekte* bewusst ausgenutzt. Anweisungen verändern jedoch nicht nur den Programmzustand, sondern werden auch von diesem beeinflusst. Daher liefert eine Anweisung trotz identischer Eingangsparameter bei wiederholter Ausführung gegebenenfalls unterschiedliche Ergebnisse. Aus diesem Grund kann die Reihenfolge von Anweisungen in einer imperativen Umgebung nicht verändert werden (Wagenknecht, 2004, Kap. 8).

Strukturiertes Programmieren: In diesem Teilbereich der imperativen Programmierung werden insbesondere die Konstrukte *Folge*, *Auswahl* und *Wiederholung* genutzt. Die

Auswahl wird in der strukturierten Programmierung durch if else-Ausdrücke realisiert, die Wiederholung durch do while. Die genannten Konstrukte, eine einhergehende Modularisierung sowie der Verzicht auf goto-Anweisungen erhöhen die Verständlichkeit und Wartbarkeit des Programmcodes (Hunt, 1979).

Prozedurales Programmieren: Die Bezeichnung wird teilweise gleichbedeutend zum imperativen Programmieren verwendet, kann aber auch einen Ansatz beschreiben, der auf der Verwendung von Prozeduren, Funktionen und komplexen Datenstrukturen beruht. Zum Aufbau eines Programms werden außerdem hierarchische Strukturen und Modularisierungstechniken eingesetzt (Nash et al., 2009; Stroustrup, 2012).

Objektorientiertes Programmieren: Die Objektorientierung basiert auf den Konzepten der *Kapselung* (engl. *encapsulation*), der *Vererbung* (engl. *inheritance*) und der *Vielgestaltigkeit* (engl. *polymorphism*). Objekte sind kapselnde Einheiten, Instanzen einer Klasse und kommunizieren untereinander. Die Klassendefinition gibt die Fähigkeiten der aus ihr erstellbaren Instanzen vor. Insbesondere besitzt ein Objekt einen Zustand und *Methoden*, um auf Nachrichten zu reagieren. Nach dem Prinzip der Kapselung kann der Objektzustand nur durch Methoden beeinflusst werden. Übergeordnete Klassen geben das allgemeine Verhalten ihrer Instanzen wieder und verhalten sich demnach als *Generalisierungen*. Anstatt verwandte Klassen grundlegend neu zu entwickeln, können diese Fähigkeiten von Elternklassen erben. Zur *Spezialisierung* einer Kindklasse werden ausschließlich deren abweichende Bereiche neu definiert. Polymorphie, als drittes Konzept der Objektorientierung, bezeichnet die Fähigkeit eines Objekts, sich je nach Situation als unterschiedlicher Typ zu verhalten. Dadurch wird eine von der Typisierung unabhängigere Programmierung ermöglicht (Schiffer, 1998; Wagenknecht, 2004, Kap. 7).

Objektbasiertes Programmieren: In dieser abgeschwächten Form des objektorientierten Programmierens werden zwar Objekte und ihre Methoden genutzt, jedoch fehlen andere Charakteristika der Objektorientierung oder sie werden nur teilweise unterstützt. Dabei kann es sich beispielsweise um die strikte Kapselung des Objektzustandes handeln (Fischer & Hofer, 2011, S. 701).

6.1.2 Paradigmen visueller Sprachen

In Abschnitt 6.3.2 diese Kapitels wird eine graphische Schnittstelle zur Definition von Anfragen in QL4BIM vorgestellt. Die genutzte Notation beruht auf dem Konzept der visuellen Programmierung, das in Myers (1990, Kap. 2.3) wie folgt definiert wird:

> „'Visual Programming' (VP) refers to any system that allows the User to specify a program in a two-(or more)-dimensional fashion. ... , conventional textual langua-

ges are not considered two dimensional since the compilers or interpreters process them as long, one-dimensional streams.“

Visuelle Sprachen werden mitunter zur Lösung ingenieurtechnischer Fragestellungen eingesetzt. Eine zentrale Entwicklung im BIM-Bereichen ist die *Dynamo*-Umgebung. Diese ermöglicht die Steuerung einer kommerziellen BIM-Anwendung. Abbildung 6.2 zeigt eine Dynamo-Programmierung zur Generierung eines parametrischen Bauteils (Oberbichler et al., 2015). Aus dem GIS-Bereich sind der *ModelBuilder* in *ESRI ArcGIS* und der *Workflow Designer* in *Autodesk Map 3D* zu nennen. Durch diese Schnittstellen können Geodatenanalysen auf Basis visueller Notationen erstellt werden (Dobesova, 2011).

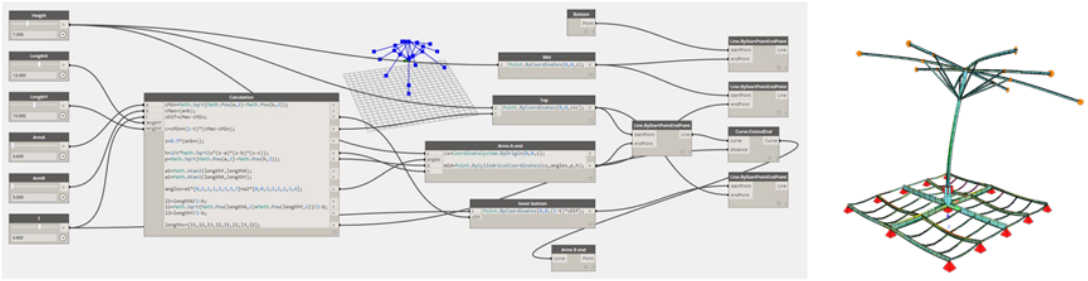


Abbildung 6.2: Dynamo-Programmierung zur Generierung eines parametrischen Bauteils, aus (Oberbichler et al., 2015)

Neben den genannten Beispielen existiert eine Vielzahl weiterer Sprachen, die zum Teil von Myers (1990) und Johnston et al. (2004) ausführlicher beschrieben werden. Eine Betrachtung visueller Sprachen in der Bauinformatik ist in (Ritter et al., 2015) enthalten.

Im Folgenden wird auf unterschiedliche Paradigmen in der visuellen Programmierung eingegangen. Eine entsprechende Kategorisierung wurde in (Schiffer, 1998, Kap. 5) unter Bezug auf (Burnett & Baker, 1994) entwickelt. Diese Kategorisierung wird hier zusammengefasst wiedergegeben.

Steuerflussorientierte Systeme: Derartige Systeme überführen das imperative Programmierparadigma in die visuelle Programmierung. Durch explizit anzuordnende Befehlselemente wird der Steuerfluss des Programms ausgedrückt. Im Detail kann die Definition des Flusses auf Basis von Anweisungssequenzen (*Flußdiagramme*, *Nassi-Shneiderman-Diagramme*), *Komponentennetzen* (*Kanalbasierter Tokenaustausch*) und *Transitionsnetzen* (*Zustandsdiagramme*, *Petrinetze*) erfolgen.

Datenflussorientierte Systeme: Diese Systeme sind mit funktionsorientierten Ansätzen verwandt und bestimmen keine strikte Befehlsreihenfolge wie steuerflussorientierte Systeme. Stattdessen zeigt die Verfügbarkeit von Daten, wann welche Operation ausgeführt

wird. Datenflussprogramme können als gerichteter Graph repräsentiert werden. Dabei kommen drei Arten von Knoten zum Einsatz: Datenquellen, Datensenken und Operationen. Kanten ermöglichen Übertragung von Daten zwischen den unterschiedlichen Knoten und werden daher als *Datenkanäle* bezeichnet. Knoten besitzen Eingänge und Ausgänge, die als Anfangs- bzw. Endpunkt für Kanten dienen. Der sich ergebende gerichtete Graph spiegelt daher nur implizit die Reihenfolge von Operationen wieder. Primär zeigt er die Datenabhängigkeiten zwischen Knoten auf. Operationen werden dann ausgeführt, wenn alle benötigten Datenquellen verfügbar sind. Operationen, die nicht voneinander abhängen, können parallel ausgeführt werden, wenn die genutzten Operationen keine Nebeneffekte beinhalten. Des Weiteren werden in diesem Ansatz standardmäßig keine Variablen genutzt.

Funktionsorientierte Systeme: In diesen Systemen wird die Einhaltung des funktionalen Programmierparadigmas strikter gehandhabt als in den verwandten datenflussorientierten Ansätzen. Dazu müssen weitere Konzepte eingeführt werden. Darunter fallen *Funktionen höherer Ordnung*, *polymorphe Funktionen* und ein implizites Typsystem. Funktionen höherer Ordnung beschreiben die Nutzung von Funktionen als Argumente weiterer Funktionen. Polymorphe Funktionen erlauben entweder Parameter unabhängig ihres Typs auf gleiche Weise zu verarbeiten oder für unterschiedlich typisierte Parameter jeweils eine Überladung bereitzustellen. Ein implizites Typsystem basiert auf einer automatischen Ableitung von Variablentypen. Zur graphischen Repräsentation von funktionsorientierten Programmen können Funktionsdiagramme dienen. Ein derartiges Diagramm stellt aus mathematischer Sicht einen gerichteten Graphen dar. In diesem sind Parameter, Ergebnisse und Funktionen als Knoten vorgehalten. Kanten dienen als Kanäle zur Werteübertragung zwischen diesen Elementen.

Objektorientierte Systeme: Wie bereits diskutiert, stellen Objekte Instanzen von Klassen dar. In entsprechenden visuellen Systemen wird häufig auf eine vorgefertigte Klassenbibliothek zurückgegriffen, auf deren Basis Instanzen erzeugt werden können. Zur Applikationsentwicklung werden graphische Repräsentationen miteinander in Beziehung gesetzt. Dem objektorientierten Paradigma folgend, kann dadurch eine Kommunikation zwischen Instanzen auf Basis von Nachrichten stattfinden. Dies ermöglicht das Lesen und Ändern von Attributen sowie den Aufruf von Methoden. Visuelle Systeme realisieren in der Regel nicht alle Hauptmerkmale der objektorientierten Programmierung. Weil mit Instanzen anstelle von Klassen gearbeitet wird, ist die Definition von Methoden sowie die Nutzung von Vererbungsbeziehungen und Polymorphismus in der Regel nicht möglich.

Constraintorientierte Systeme: In diesem Ansatz wird auf algorithmische Programmkonstrukte, wie Wertzuweisungen, Prozeduren und Ablaufstrukturen verzichtet. Ein Programm ergibt sich stattdessen durch Bedingungen (engl. constrains) zwischen Werten

bzw. Variablen. Diese müssen ständig erfüllt sein. Ein *constraint solver* findet unter Berücksichtigung der Bedingungen und vorgegebener Konstanten eine zulässige Belegung aller Variablen. Die methodische Basis derartiger Systeme stammt aus dem *constraint logic programming* (Jaffar & Lassez, 1987).

Transformationsorientierte Systeme: Derartige Systeme basieren auf *Graphgrammatiken* bzw. *Graphtransformationssystemen*. Wie textuelle Grammatiken zulässige Sätze über einem textuellen Alphabet bestimmen, dienen Graphgrammatiken zur Definition valider Graphen. Zur strukturierten Veränderung von Graphen haben sich Graphtransformationssysteme (engl. *graph rewriting systems*) etabliert. Programme können in derartigen Systemen als Graphen und angewandte Transformationen repräsentiert werden.

Beispielorientierte Systeme: Derartige Systeme existieren in zwei Ausprägungen. In der als *programming with examples* bezeichneten Variante speichert das System visuell vorgeführte Einzelschritte als eine Operation ab. Nachfolgend kann diese Operation aufgerufen werden. Die Generalität dieses Verfahrens ist gering. So verhindern Änderungen zur Aufzeichnungssitutation, wie beispielsweise die Kardinalität einer Selektion, den Operationsaufruf oder führen zu Fehlinterpretationen. Das zentrale Beispiel für das beschriebene Vorgehen ist ein *Makrorekorder*. Die zweite Variante beispielorientierter Systeme wird als *programming by examples* bezeichnet. Hier erfolgt eine Verallgemeinerung der aufgezeigten Einzelschritte. Dazu muss das System in der Lage sein, auf die Bedeutung des Gezeigten zu schließen und entsprechenden Programmcode zu erzeugen. Derartige Ansätze sind im Bereich der *künstlichen Intelligenz* anzusiedeln.

Formularorientierte Systeme: Programme werden in diesem Ansatz durch das Bearbeiten von Formularen bzw. Tabellen erstellt. Dabei können sowohl Formeln als auch skalare bzw. strukturierte Daten hinterlegt werden. Formeln können weitere Formularelemente referenzieren, deren Werte somit als Eingangsdaten dienen. Bei der Änderung von Werten wird automatisiert eine Neuberechnung abhängiger Elemente gestartet. Da derartige Abhängigkeiten intern erkannt werden, kann auf die Einbindung von Kontrollstrukturen verzichtet werden. Formal stellt das Vorgehen daher eine deklarative Programmierung dar.

6.2 Sprachkonzepte von QL4BIM

BIM-Daten werden in Bauprojekten in einer weitreichenden Wertschöpfungskette erstellt und bearbeitet. Wie gezeigt, ist es Fachanwendern ohne Informatikkenntnisse derzeit praktisch nicht möglich, die Datenbasis eines Gebäudemodells angemessen zu analysieren. Das primäre Ziel dieser Arbeit ist die Realisierung eines diesbezüglichen Ansatzes auf Basis einer formalen

Sprache. Im Folgenden werden Anforderungen an die zu entwickelnde BIM-Anfragesprache identifiziert.

- Die Sprache muss für Fachanwender mit keinen oder minimalen Programmierkenntnissen mit geringem Aufwand anwendbar sein.
- Der Ansatz muss konform zu den Prinzipien weitverbreiteter Hochsprachen wie Java, C#, C++, Python sein.
- Als Eingangsdaten sind offene Datenmodelle zu wählen. Die Entitäten dieser Modelle werden schemakonform abgebildet und bleiben stets unverändert.
- Auswertungen von BIM-Daten führen zur Betrachtung von Entitätsmengen und Entitätsrelationen. Daher müssen entsprechende Datentypen in der Anfragesprache zur Verfügung stehen.

Daraus werden weitere Entscheidungen zu den allgemeinen Eigenschaften der Sprache abgeleitet.

- Die Sprache ist für die Filterung, Analyse und Verifikation von BIM-Daten ausgelegt und beruht auf einer geringen Anzahl an Syntaxregeln. Es ist keine universelle Programmier- oder Skriptsprache.
- Die Operatoren der Anfragesprache verbergen die algorithmische Komplexität. Daher sind Konstrukte wie Kontrollstrukturen nicht Teil der Sprache.
- Die betrachtete Einheit in Analysen ist eine Entität mit ihren Eigenschaften. Einfache Datentypen treten als Attribute von Entitäten oder als Konstanten in der Sprache auf.
- Die Sprache beinhaltet Operatoren für die Untersuchung semantischer, relationaler, räumlicher und zeitlicher Aspekte der Datenbasis und sich dadurch ergebender Relationen zwischen Entitäten.
- Entitäten zeigen polymorphe Eigenschaften. In nicht-räumlichen Operatoren verhalten sich Entitäten als Objekte mit Eigenschaften (Attributen), sind jedoch passive da sie keine Methoden zum Empfang von Nachrichten und zur Änderung des Objektzustands bereitstellen. In räumlichen Operatoren werden Entitäten als räumliche Repräsentationen angesehen und können durch geometrische Algorithmen verarbeitet werden.
- Variablen können ausschließlich Entitätsmengen und Entitätsrelationen zugewiesen werden.
- QL4BIM ist eine imperative Sprache und Anfragen werden als eine geordnete Sequenz an Aussagen realisiert. Diese Ordnung schreibt die Ausführungsreihenfolge vor.

- Benutzerdefinierte Operatoren erlauben eine Anpassung der Sprache an projektspezifische Bedingungen und für bestimmte Gewerke.
- Die Sprache lässt sich über die *Textual Notation of QL4BIM (*^{t}*QL4BIM)* und die *Visual Notation of QL4BIM (*^{v}*QL4BIM)* verwenden.

6.2.1 Grundprinzip der Sprache

Dieser Abschnitt erläutert das Grundprinzip der Anfragesprache anhand der textuellen Notation ^{t}QL4BIM. Dazu werden Teile der Grammatik in Form von *Syntaxdiagrammen* (engl. *railroad diagrams*) präsentiert (Braz, 1990). Auf die visuelle Notation ^{v}QL4BIM wird in Abschnitt 6.3.2 eingegangen. Das grammatikalische Prinzip ist in beiden Notationen identisch. Abbildung 6.3 zeigt vier Syntaxdiagramme mit zentralen Elementen der Anfragesprachen.

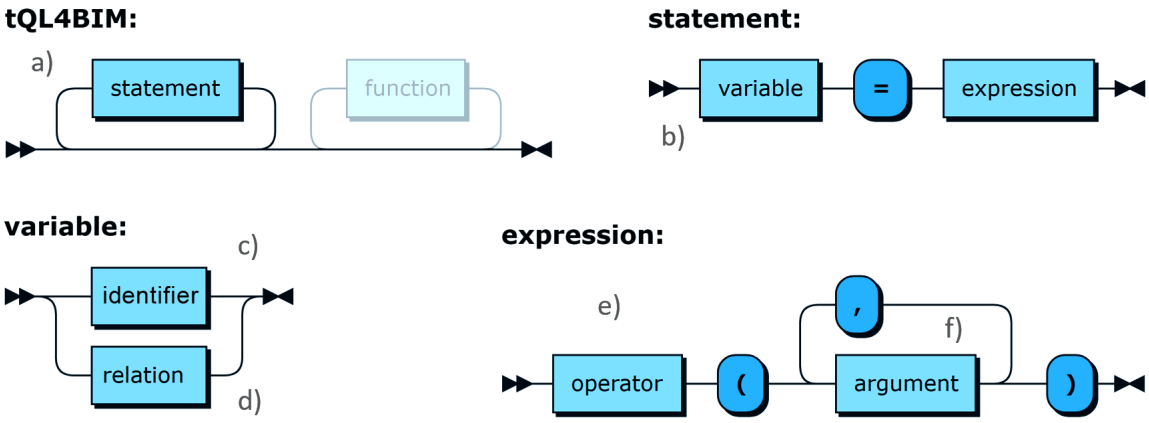


Abbildung 6.3: Die syntaktischen Grundelemente von QL4BIM

In ^{t}QL4BIM wird eine Anfrage als eine Reihe von Aussagen (engl. *statements*) definiert (siehe Abbildung 6.3a). Jede Aussage beinhaltet die Zuordnung einer Variablen an das Ergebnis eines Ausdrucks (engl. *expressions*) (b). Variablen stellen dabei Entitätsmengen (c) und Entitätsrelationen (d) dar. Als Übergabeparameter bilden sie die aktuelle Datenbasis für einen Ausdruck. Jeder Ausdruck beinhaltet den Aufruf eines Operators, der wohldefinierte Analysefunktionalität für BIM-Daten bereitstellt (e). Bei Aufruf eines Operators wird mindestens ein Argument übergeben (f).

Zur Konkretisierung des Konzepts wird Musteranfrage 4 in QL4BIM umgesetzt. Die Anfrage als Langtext lautet: „Selektiere Paare aus Deckenplatte und Wand. Dabei sollen nur Paare berücksichtigt werden, in denen sich Deckenplatte und Wand berühren und sich die Wand komplett unterhalb der Deckenplatte befindet.“. Auflistung 6.1 zeigt Musteranfrage 4 in ^{t}QL4BIM.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities is IfcSlab)
3 walls = TypeFilter(entities is IfcWall)
4 a[slab|wall] = Touches(slabs, walls)
5 b[slab|wall] = BelowStrict(a)
```

Auflistung 6.1: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ^{t}QL4BIM

In der Umsetzung der Musteranfrage ist jeweils eine Aussage pro Zeile angegeben. Die Zuweisung des Ausdrucks an eine Variable erfolgt textuell durch das =-Zeichen. In der Musteranfrage werden sowohl mengenbasierte als auch relationale Variablen verwendet. So stellen entities, slabs und walls Mengen dar (Z. 1-3). Die Bezeichner a[slab|wall] und b[slab|wall] repräsentiert hingegen zwei relationale Variablen (Z. 4 u. 5). In der Anfrage werden die Operatoren ImportModel, TypeFilter, Touches und BelowStrict eingesetzt. In den Typfiltern und dem Touches-Operator werden mengenbasierte Variablen als Übergabeparameter genutzt (Z. 2 - 4). Der BelowStrict-Operator in Zeile 5 wird mit einer relationalen Variable aufgerufen.

Folgende Regel gelten für die Anfragendefinition in QL4BIM. Die ersten drei Regeln werden bereits durch die textuelle und visuelle Grammatik umgesetzt. Regeln 4 und 5 müssen durch Komponenten des Laufzeitsystems der Sprache sichergestellt werden.

1. Mengenbasierte und relationale Variablen sind die einzigen Variablentypen. Alle anderen Typen in der Sprache werden explizit angegeben und als Konstanten und in Prädikaten verwendet.
2. Die Zuordnung eines Ausdrucks an eine mengenbasierte oder relationale Variable wird erzwungen.
3. Operatoren können nicht direkt ineinander verschachtelt werden.
4. Variablen, die als Operatorargumente dienen, müssen durch vorgehende Operationen initialisiert worden sein.
5. QL4BIM unterstützt einen indexbasierten numerischen Zugriff auf Relationsattribute. Außerdem können textuelle Bezeichner für Relationsattribute eingeführt werden. Werden diese genutzt, müssen die Bezeichner innerhalb einer Relation eindeutig sein.

Dadurch kann eine erste Einordnung von QL4BIM erfolgen.

QL4BIM ist eine imperative Sprache. In dieser wird eine Anfrage als eine strikte Folge von Aussagen definiert. Variablen werden durch ihre erstmalige Nutzung implizit deklariert und an das Ergebnis des jeweiligen Ausdrucks gebunden. Die mengenbasierte

bzw. relationale Typisierung der jeweiligen Variable kann ohne weitere Angabe aus der textuellen Repräsentation abgleitet werden. Operatoren konsumieren bereits verfügbare Variablen. Durch die Übergabe einer Variable wird der formale Parameter eines Operators mit Eingangsdaten belegt. Neben mengenbasierten und relationalen Variablen erlauben weitere Arten an Argumenten die Steuerung des Operatorverhaltens. Dazu gehören Konstanten einfacher Datentypen und Prädikate.

Nachdem das Grundprinzip erläutert wurde, folgt nun eine detailliertere Diskussion der Anfragesprache. Diese umfasst unter anderem die Typisierung der Sprache, die Überladung von Operatoren und benutzerdefinierte Operatoren. Außerdem werden der Geltungsbereich von Variablen, die Unterstützung einer variablen Anzahl an Operatorparametern und der Abstraktionsgedanke hinter QL4BIM-Operatoren besprochen.

6.2.2 Typisierung

Für die Typisierung in QL4BIM ist eine Besonderheit gegenüber anderen Sprachen relevant. So agiert QL4BIM auf Typen aus externen Schemata. Diese externen Typen werden in das spracheigene System eingebettet. Die in QL4BIM verwendete Typisierung lässt sich in drei Bereiche gliedern, die in Abbildung 6.4 dargestellt sind.

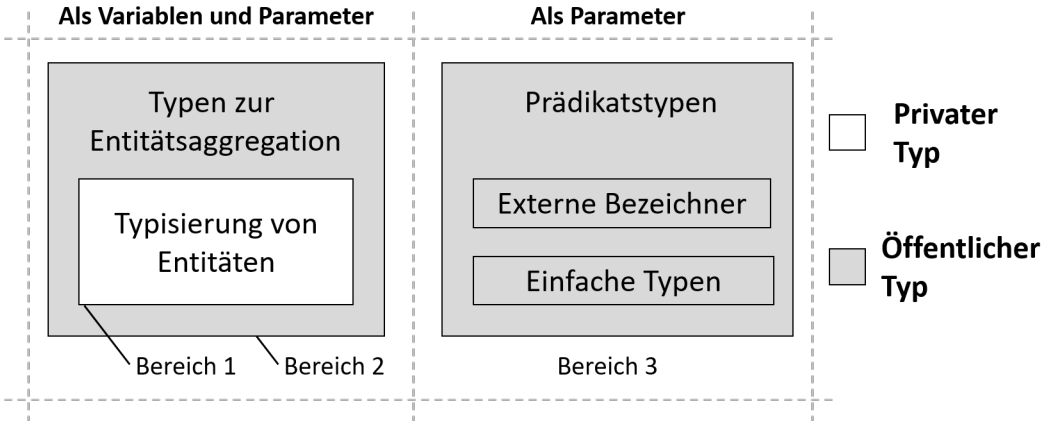


Abbildung 6.4: Die Typisierung von Variablen und Parametern in QL4BIM

Typen des ersten Bereichs dienen zur internen Vorhaltung von extern definierten Entitäten und ihrer Attribute. Der eigentliche Entitätstyp ist nicht öffentlich, da Entitäten stets in Containern vorgehalten werden. Die öffentlichen Variablentypen zur Aggregation von Entitäten werden innerhalb des zweiten Bereichs bereitgestellt. Der dritte Bereich des Typsystems bezieht sich auf die Parameterübergabe an Operatoren. In diesem werden externe Bezeichner, also Typ- und Attribut-Benennungen aus externen Schemata vorgehalten. Außerdem beinhaltet Bereich 3 einfache Typen wie Ganz- und Fließkommazahlen sowie Zeichenketten. Einfache

Typen und externe Bezeichner können alleinstehend oder in Prädikaten eingebettet als Parameter verwendet werden. Tabelle 6.1 zeigt alle in der Anfragesprache enthaltenen Typen zuzüglich einer grundlegenden Beschreibung.

Verwendung	Typ (Abkürzung)	Beschreibung
indirekt	ExternalEntity	in Mengen und Relationen enthalten
als Variable und Argument	Set	mengenbasierte Repräsentation
	Relation	relationale Repräsentation
Als Argument	RelationalAttribute (RelAtt)	Relationsattribut
	Number	Integerzahl
	Float	Fließkommazahl
	String	Zeichenkette
	Logical	Werte der ternären Logik
	Predicate	Prädikat zur Auswertung von Entitätseigenschaften und Typzugehörigkeiten
	ExternalType	Typbezeichnung aus einem externen Schema
	ExternalAttribute	Attributbezeichnung (Entitätseigenschaft) aus einem externen Schema

Tabelle 6.1: Überblick über das Typsystem von QL4BIM

Typisierung von Entitäten und ihrer Attribute

Entitäten stellen in QL4BIM komplexe Objekte externer Schemata dar. Der hier aufgezeigte Entwicklungsstand der Anfragesprache unterstützt das IFC- und das CityGML-Schema. Für jedes genutzte Schema muss festgelegt werden, in welcher Granularität dessen Elemente vorgehalten werden. Im Falle von IFC-Entitäten erfolgt eine direkte Zuordnung zu QL4BIM-Entitäten. Für Entitäten des CityGML-Schemas gilt, dass komplexe XML-Objekte als QL4BIM-Entitäten abgebildet werden. Eine externe Entität wird als ExternalEntity bezeichnet. Der Typ ist für die Vorhaltung von Attributen und Typinformationen **einer** Entität ausgelegt (vgl. Abbildung 6.5). Der Endnutzer der Anfragesprache interagiert nur indirekt mit ExternalEntitys, da diese in mengenbasierte bzw. relationale Typen eingebettet werden.

Die Typisierung von Attributen erfolgt auf Basis des verwendeten Schemas. Neben der Abbildung von Referenzbeziehungen zwischen Entitäten wird die typgerechte Unterstützung von Vergleichsoperatoren angestrebt.

Das Vorgehen wird am Beispiel des *OverallHeight*-Attributs der IfcDoor-Entität erläutert. Auflistung 6.2 zeigt die relevanten Ausschnitte des IFC-Schemas. Um einen geeigneten Typ für die Vorhaltung des Attributs auszuwählen, wird die Vererbungshierarchie der Attributtypsierung aufgelöst. Im konkreten Fall wird von IfcPositiveLengthMeasure über IfcLengthMeasure bis zu

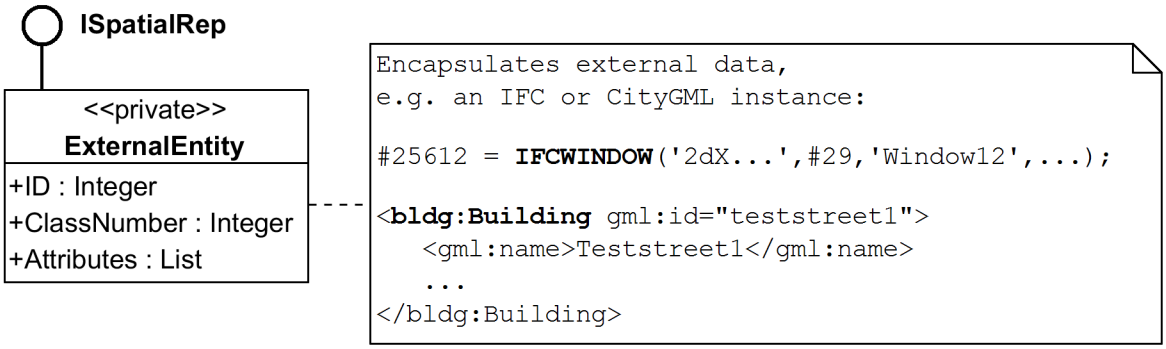


Abbildung 6.5: Die private Klasse ExternalEntity (UML-Klassendiagramm)

REAL traversiert. Somit kann der interne Typ Float, der mit dem REAL-Typ der EXPRESS-Sprache korrespondiert, zur Vorhaltung von *OverallHeight* gewählt werden.

```
ENTITY IfcDoor ...
 OverallHeight : OPTIONAL IfcPositiveLengthMeasure; ...
END_ENTITY;
TYPE IfcPositiveLengthMeasure = IfcLengthMeasure;
 WHERE WR1 : SELF > 0.;
END_TYPE;
TYPE IfcLengthMeasure = REAL;
END_TYPE;
```

Auflistung 6.2: Traversierung eines externen Schemas zur Typisierung von Enitätsattributen

Die benötigte Zuordnung einfacher EXPRESS- und CityGML-Typen zu einfachen QL4BIM-Typen ist in Tabelle 6.2 dargestellt.

EXPRESS-Typ	CityGML-Typ	QL4BIM-Typ
REAL	double / float	Float
INTEGER	integer	Number
STRING	string	String
LOGICAL	kein einfacher, ternärer Typ vorhanden	Logical
BOOLEAN	boolean	Logical

Tabelle 6.2: Zuordnung einfacher EXPRESS- und CityGML-Typen zur einfachen QL4BIM-Typen

Durch die beschriebene Funktionalität des ExternalEntity-Typs wird eine Analyse auf Basis von Typzugehörigkeiten, Attributwerten und Referenzbeziehungen ermöglicht. Da QL4BIM für die Analyse einer räumlichen Datenbasis ausgelegt ist, weist der entwickelte Entitätstyp polymorphes Verhalten auf. Er wandelt sich bei Bedarf implizit in eine räumliche Repräsentation um. Möglich wird dies durch die Implementierung des ISpatialRep-Interfaces durch ExternalEntity (siehe Abbildung 6.6).

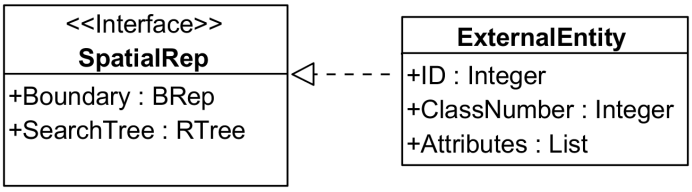


Abbildung 6.6: Die Klasse ExternalEntity kann als semantische, und über das Interface ISpatialRep, als räumliche Repräsentation interpretiert werden (UML-Klassendiagramm)

Durch eine Vorprozessierung beim Import wird für jede Entität mit Geometriedaten ein BRep aus der Datenbasis gewonnen und an das Attribut *Boundary* gebunden. Um die Zuordnung zwischen BRep und semantischer Entität durchzuführen ist im Falle der IFC eine Traversierung mehrerer Klassen nötig (siehe Abbildung 3.7). In CityGML ist die Zuordnung durch die in semantischen Klassen inkludierte Geometriebeschreibung mit geringerem Aufwand zu bewerkstelligen.

Da beide Datenformate für eine Entität mehrere räumliche Repräsentation unterstützen, stellt sich die Frage mit welchen räumlichen Daten eine semantische Entität verknüpft werden soll. Auf Grund der entwickelten räumlichen Operatoren und deren Auslegung auf umfangreiche räumliche Datenbestände wird hier davon ausgegangen, dass stets die detailreichste Variante gewählt werden kann. In CityGML ist dies das BRep des höchsten verfügbaren LoD, in IFC die detaillierteste IfcGeometricRepresentationItem.

Wie beschrieben verwenden die räumlichen Operatoren eine Indizierung mit R*-Trees. Über das Attribut *SearchTree* des Interfaces kann diese Baumstruktur für jede Entität mit geometrischen Daten abgerufen werden.

Typen zur Entitätsaggregation

Aus den Musteranfragen in Kapitel 4.2 lassen sich Rückschlüsse für die benötigten Variablentypen in QL4BIM ziehen. Zum einen wird hinsichtlich der Ergebnismultiplizität eine mengenbasierte Vorhaltung benötigt. In Anfragen werden außerdem Beziehungen die zwischen Entitäten gelten, identifiziert. Um diese Zusammenhänge beizubehalten und in nachgeordneten Ausdrücken weiterzuverarbeiten, ist ein relationaler Datentyp erforderlich.

Der relationale Datentyp ermöglicht es containertypisierte Attribute aufzulösen und als 1:1-Beziehungen vorzuhalten. Auflistung 6.3 zeigt diesbezüglich die EXPRESS-Spezifikation von IfcRelContainedInSpatialStructure. Durch die SET-Typisierung im *RelatedElements*-Attribut wird eine 1:n-Beziehung zwischen räumlicher Struktur und enthaltenen Bauteilen aufgebaut. Der QL4BIM-Typ Relation ermöglicht es, eine für Auswertungen vorteilhafte 1:1-Beziehung zu gewinnen. Tabelle 6.3 zeigt eine entsprechende Vorhaltung von räumlichen Strukturen und Bauteilen.

```
1 ENTITY IfcRelContainedInSpatialStructure
2  SUBTYPE OF (IfcRelConnects);
3   RelatedElements : SET [1:?] OF IfcProduct;
4   RelatingStructure : IfcSpatialElement;
5 END_ENTITY;
```

Auflistung 6.3: 1:n-Beziehung zwischen räumlicher Struktur und IfcProducts

pair[structure|product]	
spatialEntity1	buildingElement4
spatialEntity1	buildingElement8
spatialEntity2	buildingElement2
spatialEntity2	buildingElement6
spatialEntity2	buildingElement9

Tabelle 6.3: Beispielhafter Inhalt der QL4BIM-Variable pair[structure|product] vom Typ Relation, Auflösung einer EXPRESS-basierten 1:n-Beziehung

Die Klasse Set erlaubt in QL4BIM die mengenbasierte Aggregation von Entitäten und beinhaltet daher grundsätzlich keine Duplikate. Eine relationale Variable vom Typ Relation besteht aus einer Menge an Tupeln, die wiederum Entitäten enthalten. Eine einzelne Entität bzw. ein einzelnes Tupel wird in QL4BIM als Menge mit einem Element bzw. als Relation mit einem Tupel vorgehalten. Wie ExternalEntity ist auch Tuple ein privater QL4BIM-Typ. Beide Variablentypen und ihr Zusammenspiel mit der ExternalEntity-Klasse sind in Abbildung 6.7 dargestellt.

Ein Attribut einer Relation korrespondiert mit einem bestimmten Tupelindex. In QL4BIM verweist ein Attribut daher auf die Entitäten aller Tupeln an dem entsprechenden Index. Relation unterstützt sowohl anonyme als auch benannte Relationsattribute. Benannte Attribute können durch ihren Bezeichner und ihren 1-basierten Index angesprochen werden. Anonyme Relationsattribute können ausschließlich über ihren Index referenziert werden. Relationen mit benannten Attributen geben über die Anzahl an Bezeichnern die Stelligkeit der Relation an. Auf die Angabe der Stelligkeit bei Relationen mit anonymen Attributen kann verzichtet werden. Als Relationsrepräsentation beinhaltet eine Relation-Instanz keine identischen Tupel.

Grammatikalisch stellen Set-Variablen einfache Bezeichner dar. Relationsvariablen vom Typ Relation werden hingegen durch die Verwendung eines Relationsbezeichners in Kombination mit eckigen Klammern erstellt. Sollen die Attribute der Relation benannt werden, müssen alle Attributbezeichner innerhalb der Klammern aufgeführt werden. Als Trennung dient das |-Zeichen (engl. *vertical bar*, *pipe*). Auflistung 6.4 zeigt Beispiele für die Verwendung des Set- und Relation-Typs als Variablen. Abbildung 6.8 verdeutlicht die Datenstrukturen.

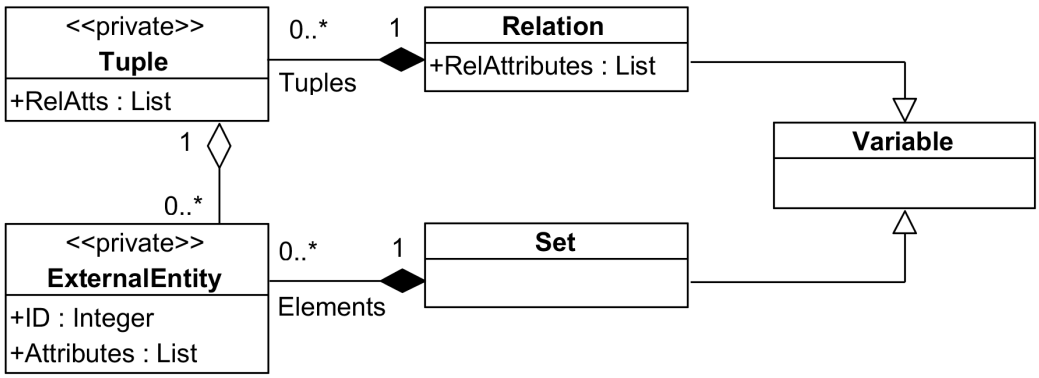


Abbildung 6.7: Mengen und Relationen in QL4BIM: der Set-Typ und der Relation-Typ (UML-Klassendiagramm)

```
1 entities = ...  //Set variable
2 touchingElements[] = ...  //Relation variable with anonymous attributes
3 touchingElements[wall|door] = ...  //Relation variable with named attributes
```

Auflistung 6.4: Beispiele zur Verwendung von Set- und Relation-Variablen

entities
entity#101
entity#202
entity#103
...

touchingElements	
entity#102	entity#201
entity#101	entity#211
entity#104	entity#221
...	...

touchingElements	
wall	door
entity#102	entity#201
entity#101	entity#211
...	...

Abbildung 6.8: Visualisierung der Strukturen von Set- und Relation-Variablen

Typisierung in Parameterübergaben

Operatoren in QL4BIM stellen Analysefunktionen für IFC- und CityGML-Daten bereit. Zur Übergabe eines zu verarbeitenden Datenbestands und für die Funktionssteuerung werden die formalen Parameter einer Operation mit konkreten Argumenten belegt.

Zur Übergabe des zu verarbeitenden Datenbestands werden die bereits diskutierten Typen Set und Relation genutzt. Zur Auswahl von Attributen einer übergebenen Relation dient der RelationalAttribute-Typ. Dieser repräsentiert einen 1-basierten Index als Integerwert. Alternativ kann der in der impliziten Variablendeklaration genutzte Attributbezeichner angegeben werden.

Die Verarbeitung und Vorhaltung relationaler Daten stellen in QL4BIM ein fundamentales Konzept dar. Folgende Möglichkeiten bestehen in der Anfragesprache, wenn eine relationale Variable als Argument an einen Operator übergeben wird.

1. Operatoren erwarten eine oder mehrere Relationen ohne Auswahl eines speziellen Relationsattributs. Dann wird Relation als Parametertyp genutzt.

2. Falls ein Operator eine Relation verarbeitet und zusätzlich die Auswahl von Relationsattributen notwendig ist, wird Relation in Kombination mit RelationalAttribute verwendet.

3. Im Allgemeinen wird bei der Übergabe einer Relation mit Attributauswahl die gesamte Ursprungsrelation zurückgegeben und nicht nur die ausgewählten Attribute. Somit bleiben gewonnene Relationen trotz nachfolgender Verarbeitungsschritte erhalten.

4. Werden bestimmte Relationsattribute in einer Anfrage nicht mehr benötigt, können sie über einen Projektionsoperator entfernt werden.

Neben der Übergabe von Mengen und Relationen und der Auswahl von Relationsattributen muss zur Steuerung von Operatoren auch die Auswahl eines bestimmten Entitätsattributs möglich sein. Die geschieht in QL4BIM über einen *Attributzugriff* (engl. *attribute access*). Dieses Konstrukt kann sowohl bei mengenbasierten als auch relationalen Parametern verwendet werden. Auflistung 6.5 zeigt unterschiedliche Varianten des Zugriffs.

```
1 ... = OperatorA(entities.Name = "FirstFloor")
2 ... = OperatorA(a, a[wall].Name = "FirstFloor")
3 ... = OperatorA(a, [wall].Name = "FirstFloor")
```

Auflistung 6.5: Unterschiedliche Varianten des Attributzugriffs in QL4BIM

Erfolgt ein Attributzugriff auf einen Set-Parameter wird diesem ein externes Attribut mittels Punkt-Operator angefügt (Z. 1). Im Falle eines mengenbasierten Parameters wird zum Attributzugriff ein weiterer Parameter vom Typ RelationalAttribute benötigt. Der entsprechende Bezeichner wird hier in eckige Klammern eingebettet (Z. 2). Die explizite Wiederholung des relationalen Bezeichners ist optional (Z. 3). Wenn mehrere Entitätsvariablen übergeben werden, kann aber die Wiederholung des Relationsbezeichners sinnvoll und durch das Laufzeitsystem der Sprache gefordert werden. Dadurch wird klargestellt auf welche Relation sich der jeweilige Zugriff bezieht. Auf die formalen Regeln zum attribute access wird in Abschnitt 6.3.1 näher eingegangen.

Im Folgenden wird die Verwendung von Relationen als Variablen und Parameter erläutert. Auflistung 6.6 zeigt dementsprechende Beispielaussagen. Die eigentliche Semantik der Anfrage wird an dieser Stelle nicht betrachtet. In der ersten Zeile wird die zweistellige relationale Variable `a` implizit deklariert und nachfolgend dem `Deassociater`-Operator übergeben. Das Argument für den Attributzugriff `[element].Decomposes` beinhaltet wie zuvor erläutert ein Relationsattribut und ein Entitätsattribut. Somit sind sowohl der Tupelindex als auch das auszuwertende Entitätsattribut bekannt (Z. 2). Die dreistellige Variable `c` in Zeile 3 zeigt, dass die ursprünglichen Attribute der Eingangsrelation `b` erhalten bleiben. Im Fall des `Difference`-

Operators werden relationale Variablen ohne weitere Attributzugriffe übergeben. Das Ergebnis des Ausdrucks wird an die Variable `d` mit anonymen Relationsattributen gebunden (Z. 4).

```
1 a[element|material] = ...
2 b[element|material|decompose] = Deassociater(a, [element].Decomposes)
3 c[roof|material|decompose] = TypeFilter(b, [element] is IfcRoof)
4 d[] = Difference(b, c)
```

Auflistung 6.6: Verwendung von relationalen Variablen und Parametern in QL4BIM

Zur Repräsentation einfacher Konstanten dienen in QL4BIM die Subklassen von Constant (siehe Abbildung 6.9). Diese korrespondieren mit der `simpleType`-Produktion der Grammatik und ermöglichen die Vorhaltung von Zeichenketten (String), Integerzahlen (Number), Fließkommazahlen (Float) und booleschen Werten (Logical). Diese einfachen Typen kommen als direkte Argumente in Operatoraufrufen und in den nachfolgend diskutierten Prädikatargumenten zum Einsatz. Variablen in QL4BIM sind ausschließlich für die Vorhaltung von BIM-Entitäten vorgesehen. Daher werden Constants stets als explizite Werte angegeben.

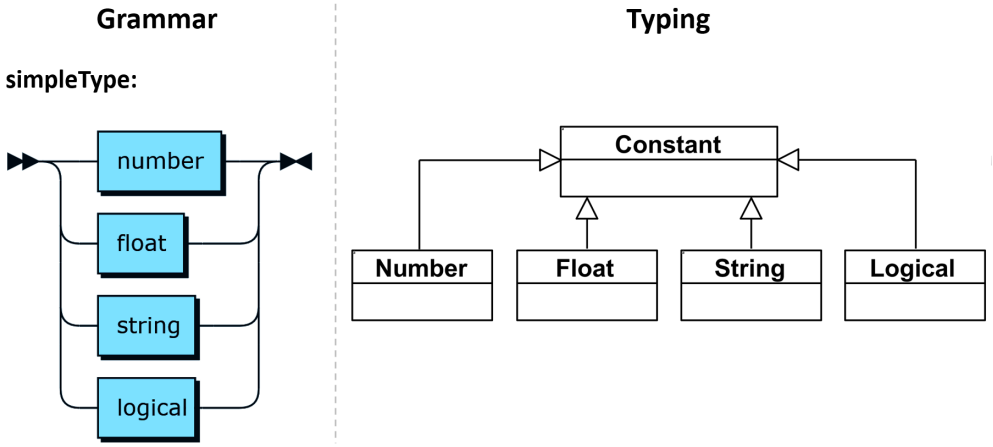


Abbildung 6.9: Grammatikalische Produktionen und Typen einfacher Konstanten

Auflistung 6.7 zeigt die Verwendung eines Strings als Übergabeparameter für den `ImportModel`-Operator. Die Zeichenkette gibt in diesem Fall die zu ladende IFC-Datei an.

```
entities = ImportModel("C:\TranslaTUM.ifc")
```

Auflistung 6.7: Einfache Typen wie String agieren in QL4BIM stets als Konstanten und werden als Literale vorgehalten

Die letzte Variante an Parameterübergaben stellen Prädikate dar. Diese existieren in QL4BIM in drei Varianten. In einem TypePredicate wird der Inhalt einer mengenbasierten oder relationalen Variable mit einem extern definierten Typ verglichen. Das Prädikat beinhaltet neben der Containervariable das `is`-Keyword und den zu prüfenden externen Typ als ExternalType-

Instanz. Der Prädikatentyp AttributePredicate dient zur Untersuchung von Entitätseigenschaften. Innerhalb des Prädikats muss die zu verarbeitende Eigenschaft durch ein Attributzugriff ausgewählt werden. Die Entitätseigenschaft (ExternalAttribute) wird mit einer Konstante (Constant) oder einer weiteren Eigenschaft verglichen. Mit einem CountPredicate kann zudem die Anzahl von Beziehungen einer Entität als Filterkriterium genutzt werden.

Auflistung 6.8 demonstriert die zwei Prädikatvarianten. In Zeile 2 wird eine mengenbasierte Typfilterung ausgeführt. Ein TypePredicate wird ebenfalls in Zeile 5 eingesetzt. Hier wird die relationale Variable `a` übergeben und deren Attribut *element1* auf Typzugehörigkeit getestet. Ein AttributePredicate wird in den `AttributeFilter`-Operatoren verwendet. In Zeile 3 wird in der Attributfilterung eine mengenbasierte Variable übergeben. In der zweiten Attributfilterung wird das *Description*-Attribut des zweiten Index (`element`) von Relation `b` untersucht (Z. 6).

```
1 entities = GetModel("C:\institute.ifc")
2 elements = TypeFilter(entities is IfcBuildingElement)
3 elements = AttributeFilter(entities.Name ~ "Trockenbau")
4 a[element1|element2] = Touch(elements, elements)
5 b[slab|element] = TypeFilter(a, [element1] is IfcSlab)
6 c[slab|element] = AttributeFilter(b, [element].Description = "Phase 3")
```

Auflistung 6.8: Beispielhafte tQL4BIM-Anfrage zur Demonstration von Prädikaten

Typisierungsgrade in QL4BIM

Das Typsystem von QL4BIM besitzt unterschiedlich stark typisierte Bereiche. Die Repräsentation von externen Entitäten ist schwach typisiert. Das bedeutet, dass alle IFC-Entity-Typen und komplexe XML-Elemente aus CityGML einheitlich über den ExternalEntity-Typ abgebildet werden können. Die öffentlichen QL4BIM-Typen Set und Relation aggregieren Entitäten. Welcher externe Typ eine ExternalEntity konkret vorhält, wird nicht betrachtet. So lassen sich Mengen und Relationen aus unterschiedlichen Typen und unterschiedlichen Schemata aufbauen.

Ebenfalls schwach typisiert sind ExternalTypes und ExternalAttributes, die zur Repräsentation von extern definierten Typbezeichnungen und Attributbezeichnungen dienen. Bis auf die zu beachtende *Zeichengrundmenge* besteht keine Einschränkung für die enthaltenen Bezeichner. Es ist Aufgabe des Laufzeitsystems auf Bezeichner, die nicht im IFC- und CityGML-Schema enthalten sind, hinzuweisen bzw. diese abzulehnen.

Im Kontrast zu dieser relaxierten Typsierung für externe Entitäten und Bezeichner sind QL4BIM-eigene und benutzerdefinierte Operatoren stark typisiert. Sie lassen sich nur mit der passenden Typisierung aufrufen. Das heißt, dass für den jeweiligen Operator eine Überladung

existieren muss, deren Signatur mit der Typisierung der übergebenen Argumente übereinstimmt.

Das zentrale Element der Untersuchung in QL4BIM ist die Entität, die durch ein externes Schema wie der IFC oder CityGML typisiert ist. Dadurch ergibt sich der Feinheitsgrad des Variablensystems der Sprache. In diesem beinhalten Variablen stets komplette Entitäten und keine einfachen Datentypen wie Zeichenketten oder Zahlenwerte. Dadurch bleiben die hoch-strukturierten Entitätsdaten erhalten und lassen sich konform zu ihren Ursprungsschemata interpretieren.

Entitäten werden in mengenbasierten und relationalen Containervariablen vorgehalten und verarbeitet. Dabei wird keine Einschränkung gegenüber externen Typen getroffen. Somit kann eine mengenbasierte Variable beispielsweise Wände und Stützen beinhalten oder Entitäten unterschiedlicher Schemata aufnehmen. Der Grund für diese Designentscheidung ist, dass eine starke Typisierung auf Basis externer Entitäten für eine Analyse als zu restriktiv eingeschätzt wird.

Der Aufruf von Operatoren basiert hingegen auf einer strikten Typisierung. So können Operatoren nur mit der passenden Signatur aus mengenbasierten, relationalen und einfachen Typen aufgerufen werden. Die starke Typisierung hinsichtlich der Operatoren hilft in der semantischen Prüfung fehlerhafte Anfragen zu identifizieren und sinnvolle Fehlermeldungen auszugeben.

6.2.3 Operatorüberladungen

Die mehrfache Deklaration identisch benannter Funktionen wird als *Überladen* (engl. *overloading*) bezeichnet. Die Funktionen unterscheiden sich dabei in ihren formalen Parametern (Fischer & Hofer, 2011, S. 932). Das Prinzip der Überladung wird in QL4BIM genutzt, um Operatoren einheitlich sowohl für mengenbasierte als auch relationale Argumente anbieten zu können. Zudem lassen sich Kontrollstrukturen abstrahieren und Standardwerte für Parameter einführen.

Zur Dokumentation der entwickelten Operatoren wird folgende Konvention für Operatorsignaturen gewählt. Die Typisierung von Argumenten und Rückgaben wird mittels *Postfix*-Schreibweise in der Operatorsignatur angegeben. Das :-Zeichen wird dabei als Trennzeichen zwischen Argument- und Typbezeichnung genutzt. Diese Darstellung dient zur Beschreibung von Operatoren und ist nicht Teil der Sprachgrammatik. Werden benutzerdefinierte Operatoren in QL4BIM entwickelt, gilt hingegen der entsprechende Teil der Sprachgrammatik.

Auflistung 6.9 enthält zwei Signaturen eines beispielhaften QL4BIM-Operators. In der ersten Überladung wird eine mengenbasierte Variable und eine Zeichenkette, in der zweiten Überladung eine Relation und eine Zeichenkette übergeben.

```
OperatorA(aSet : Set, aString: String) : Set
OperatorA(aRelation : Relation, aString : String) : Relation
```

Auflistung 6.9: Beispiel eines überladenen QL4BIM-Operators

Überladungen können sich zusätzlich, jedoch nicht ausschließlich, in ihrem Rückgabetyp unterscheiden. So liefert die erste Variante von `OperatorA` ein Set, die zweite eine Relation. Ist der Rückgabetyp eines Operators somit abhängig von der Parameteranzahl und -Typisierung lässt sich dies mit Überladungen direkt abbilden. Ein Beispiel hierfür ist der Projektionsoperator. Wird aus einer Relation nur **ein** Attribut ausgewählt, wird eine Menge zurückgegeben. Werden hingegen **mehrere** Attribute selektiert, wird eine Relation zurückgegeben.

Anfragen in QL4BIM werden ohne die explizite Nutzung von *Kontrollstrukturen* (engl. *control flow statements*) erstellt. Auch in diesem Kontext sind Überladungen vorteilhaft, da somit unterschiedliche Varianten der Kontrollstrukturen abgebildet werden können.

Generell gilt in der Anfragesprache folgende Semantik für überladene Operatoren. Werden **zwei** Entitätsvariablen übergeben, bewirkt diese eine m:n-Analyse. Dies gilt für Mengen sowie Relationen. Wird nur **eine** Entitätsvariable vom Typ Relation übergeben, wird eine 1:1-Analyse durchgeführt. Bei der Übergabe eines Sets erfolgt eine einfache Iteration über alle enthaltenen Entitäten.

Das Prinzip wird an den fünf Überladungen eines beispielhaften Operators, die in Auflistung 6.10 dargestellt sind, verdeutlicht. Die Rückgabewerte der Überladungen werden hier nicht betrachtet und sind daher nicht spezifiziert.

```
1 OperatorA(set1 : Set, set2 : Set)
2 OperatorA(relation1 : Relation, attribute1 : RelAtt,
            relation2 : Relation, attribute2 : RelAtt)
3 OperatorA(relation1 : Relation, attribute1 : RelAtt, attribute2 : RelAtt)
4 OperatorA(relation1 : Relation)
5 OperatorA(set : Set)
```

Auflistung 6.10: Fünf Überladungen eines Operators zur Abstraktion unterschiedlicher Kontrollstrukturen

Werden **zwei** Entitätsvariablen übergeben, wird eine entsprechende Produktmenge erstellt. So werden in der ersten Überladung aus allen Entitäten von `set1` und allen Entitäten von `set2` Paare gebildet und prozessiert (Z. 1). In der zweiten Überladung wird eine Produktmenge aus den Tupeln beider Relationen gebildet. Die Selektion der relevanten Entitäten aus den

Tupeln erfolgt durch die RelationalAttribute-Parameter (Z. 2). Beide Varianten stellen eine m:n-Analyse dar.

Eine 1:1-Analyse wird ausgeführt, wenn **eine** relationale Variable übergeben wird. Dies ist in den dritten und vierten Überladungen der Fall (Z. 3 u. 4). Die zu betrachtenden Entitäten werden in der dritten Überladung durch die zwei RelationalAttribute-Parameter spezifiziert. Die vierte Überladung zeigt die Verwendung einer Überladung für Standardfälle. Der Operator verhält sich identisch mit der dritten Überladung, wobei stets die ersten beiden Relationsattribute ausgewählt werden. In der letzten Überladung mit **einem** mengenbasierten Parameter erfolgt eine einfache Iteration über alle Mengenelemente (Z. 5).

Operatorüberladungen sind nötig, um die Anfragefunktionalität konsistent sowohl für Mengen als auch für Relationen bereitzustellen. Außerdem bestimmt die Parameteranzahl in Operatoren mitunter deren Rückgabetyp. In QL4BIM werden zusätzlich unterschiedliche Varianten an Kontrollstrukturen durch Überladungen abgebildet. Somit lassen sich einfache und verschachtelte For-Schleifen für mengenbasierte und relationale Parameter abstrahieren.

6.2.4 Benutzerdefinierte Operatoren in ᵗQL4BIM

Neben einer Reihe von Aussagen kann eine QL4BIM-Anfrage Operatordefinitionen enthalten. Konzeptionell werden dadurch Funktionen in die Sprache eingeführt. Diese erhöhen die Wiederverwendbarkeit von Anfrageteilen und ermöglichen gewerke- und projektspezifische Anpassungen der Sprache.

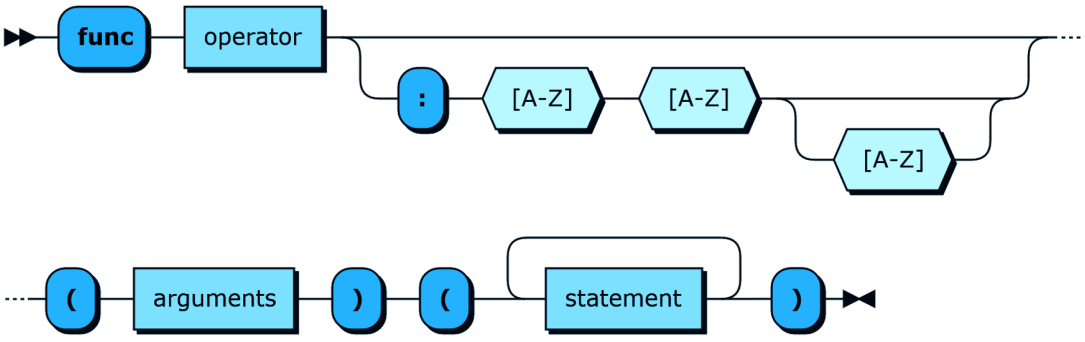


Abbildung 6.10: Produktion zur Erstellung benutzerdefinierter Operatoren

QL4BIM nutzt ausschließlich mengenbasierte und relationale Variablen. Neben relationalen Attributen sind dies die einzigen zulässigen Parameter in benutzerdefinierten Operatoren. Alle weiteren Typen müssen in derartigen Operatoren explizit angegeben werden. Dies entspricht dem Vorgehen im globalen Aussagenbereich einer Anfrage. Als Ergebnis liefert ein

benutzerdefinierter Operator die als letztes zugewiesene Variable zurück. Um mengen- und relationale Operatorvarianten sowie Standardwerte für Parameter abbilden zu können, werden in QL4BIM Überladungen auch für benutzerdefinierte Operatoren unterstützt.

Syntaktisch erfolgt die Spezifizierung der Parametertypisierung nach den folgenden drei Regeln. Mengenbasierte Parameter werden durch einfache Bezeichner ausgedrückt. Relationale Parameter werden über einen Bezeichner mit dem Postfix `[]` und enthaltenen Attributen repräsentiert. Als Rückgabetypen kommen Relation und Set in Frage, was über die Typisierung der zuletzt zugewiesenen Variable entschieden wird. Einfache Typen und relationale Attributzugriffe können nicht als Variablen übergeben werden, sondern müssen explizit innerhalb des benutzerdefinierten Operators angegeben werden.

Das Standardvorgehen zur Anfragedefinition sieht die Deklaration benannter relationaler Attribute vor. Dies dokumentiert die Absicht der Anfrage und erhöht ihre Verständlichkeit. Hingegen ermöglicht die Verwendung anonymer relationaler Attribute und numerischer Indizes in Operatoraufrufen eine kompakte Anfragedefinition. Falls diese durch den Nutzer vorgezogen wird, sind prägnante, gekürzte Operatorbezeichner ebenfalls sinnvoll. Ist ein Objekt innerhalb einer Sprache über mehrere Bezeichner ansprechbar, wird dies als *aliasing* bezeichnet. QL4BIM unterstützt die Definition eines *alias* für jeden Operator. Auflistung 6.11 enthält die Definitionen des überladenen `MyOperator`s mit dem alias `MOP`. In der ersten Überladung wird ein Set (Z. 1), in der zweiten eine Relation übergeben (Z. 7).

```
1  ... //global query scope
2
3  func MyOperator:MOP(entities)
4  (
5    entitiesA = OperatorA(entities, "aString")
6    entitiesB = OperatorB(entitiesA.anAttribute >= 5.12)
7  )
8
9  func MyOperator:MOP(a[wall|door])
10 (bbbbbbbbbb
11   b[wall|door] = OperatorA(a, [wall], "aString")
12   c[wall|door] = OperatorB(b, [door].anAttribute >= 5.12)
13 )
```

Auflistung 6.11: [t]QL4BIM-Anfrage mit einem überladenen, benutzerdefinierten Operator inklusive Alias

Relationale Parameter können nicht übergeben werden, sondern müssen explizit innerhalb des benutzerdefinierten Operators angegeben werden. Da zusätzliche relationale Attribute stets am Ende einer Relation angefügt werden, kann durch die Nutzung negativer Indizes jedoch die Prozessierung von n-stelligen Relationen verallgemeinert werden. Auflistung 6.12 enthält die Signatur eines entsprechenden Operators. Innerhalb des Funktionsblocks von `NewOperator`

wird in Zeile 3 `OperatorA` aufgerufen. Dieser Operator liefert eine in der Stelligkeit um eins erhöhte Relation. Unabhängig von der ursprünglichen Stelligkeit der Variable `a` kann mit `-1` als Index stets dieses hinzugekommene Attribut im `OperatorB` ausgewählt werden (Z. 4).

```
1 func NewOperator(a[])
2 (
3   b[] = OperatorA(a, [1].anAttribute)
4   c[] = OperatorB(b, [-1].anAttribute >= 5.12)
5 )
```

Auflistung 6.12: Negative Indizes ermöglichen in benutzerdefinierten Funktionen eine Verarbeitung unabhängig von der Eingangsstelligkeit relationaler Parameter

Benutzerdefinierte Operatoren erlauben eine Anpassung der Sprache an projekt- und domänenspezifische Gegebenheiten. An derartige Operatoren können ausschließlich Mengen und Relationen als Variablen übergeben werden. Einfache Typen müssen als Konstanten innerhalb der benutzerdefinierten Operatoren angegeben werden. Dies verringert die Flexibilität derartiger Operatoren. Als wichtiger wird hier jedoch das Ziel erachtet, die Sprache überschaubar und einheitlich zu gestalten. Durch das gewählte Vorgehen verhalten sich Operatoren stets einheitlich, unabhängig davon, ob sie im globalen Anfrageteil oder eingebettet in benutzerdefinierten Operatoren verwendet werden.

6.2.5 Variablenübergabe und Variablenscope

QL4BIM wurde als reine Anfragesprache entwickelt und beinhaltet daher keine Manipulationsfunktionalität. Da Entitäten durch Anfragen nie modifiziert werden, ist die Art der Übergabe von Variablen für diesen Gesichtspunkt irrelevant.

Während der Laufzeit einer Anfrage sind die beiden Variablentypen Set und Relation in QL4BIM jedoch veränderbar. Konzeptionell stellen die beiden Variablentypen aggregierte Verweise (engl. references) auf importierten IFC- und CityGML-Entitäten dar. Durch ihre Veränderbarkeit kann eine Variable zu verschiedenen Zeitpunkten auf unterschiedliche Mengen bzw. Relationen verweisen.

Die Modifikation einer Variablen kann ausschließlich durch Zuweisung eines neuen Ergebnisses geschehen und entspricht daher einem *reassignment*. Im Falle von Relation kann sich durch ein Reassignment die Stelligkeit der relationalen Variable ändern. Die Nutzung einer Variable als Argument in einem Operator kann keine Änderung an dieser bewirken, da Variablen als Werte (engl. *by value*) und nicht als Referenzen übergeben werden (engl. *by reference*).

In QL4BIM wird ein lexikalischer Geltungsbereich verwendet. Dies bedeutet, dass der Gültigkeitsbereich einer Variablen allein aus ihrer Lokalisierung in der Anfragedefinition ermittelt werden kann. Dies ist entscheidend, wenn Variablen an neue Werte gebunden werden. Ein Beispiel zu Geltungsbereichen und zur Parameterübergabe in einer QL4BIM-ähnlichen, imperativen Sprache zeigt Auflistung 6.13. In dieser wird der Operator `OperatorA` definiert (Z. 4-7) und aufgerufen (Z. 2). Je nachdem, welche Regeln zum Geltungsbereich und zur Parameterübergaben angewendet werden, unterscheidet sich der finale Wert der Variabel `entitiesC`. Zudem muss entschieden werden, ob innerhalb von `OperatorA` der Zugriff auf `entitiesB` erlaubt ist.

```
1 entitiesB = ...   // outside variable
2 entitiesC = OperatorA(entities) //by value parameters
3
4 OperatorA(entitiesA)(
5   entitiesB = ... // modifies outside variable, not possible in QL4BIM
6   entitiesA = ... // modifies parameter, not visible outside OperatorA
7 )
```

Auflistung 6.13: Geltungsbereiche und Parameterübergaben in QL4BIM

In QL4BIM gelten folgende Regeln für den Geltungsbereich von Variablen. Eine Anfrage definiert einen Bereich aus Statements (engl. *block*). Ein benutzerdefinierter Operator realisiert ebenso einen Block. Jeder Block ergibt einen unabhängigen Geltungsbereich. Insbesondere sind Variablen des globalen Anfrageblocks innerhalb von benutzerdefinierten und QL4BIM-eigenen Operatoren nicht sichtbar. Ein Datentransfer zwischen Blöcken kann ausschließlich durch Übergabeparameter mit by value-Semantik erfolgen.

Auflistung 6.14 zeigt eine Anfrage, in der Paare aus sich berührenden Wände und Stützen identifiziert werden (Z. 2 - 4). Anschließend werden für jede Wand alle Räume (IfcSpaces) gefunden für die die Wand eine Begrenzung darstellt (Z. 3). Der benutzerdefinierte Operator `NewOperator` führt für eine übergebene relationale Variable `rel[]` eine Projektion aus, so dass die ersten zwei Relationsattribute übernommen werden. Zeile 4 und 5 zeigen das Reassignment einer relationalen Variable inklusiv deren Stelligkeitsveränderung. So ist die Variable `a` in Zeile 4 zweistellig und beinhaltet Paare aus Stützen und Wänden. In Zeile 5 ist die Variable `b` dreistellig und enthält Stützen, Wände und Räume. Das Ergebnis des benutzerdefinierten Operators wird der Variable `c` in Zeile 6 zugeordnet. Auf Grund der Parameterübergabe by value und den Scoperegeln von QL4BIM bleibt die Variable `b` **nach Anfrageausführung** dreistellig und enthält Stützen, Wände und Räume.

```
1  entities = GetModel("C:\institute.ifc")
2  columns = TypeFilter(entities is IfcColumn)
3  walls = TypeFilter(entities is IfcWall)
4  a[column|wall] = Touches(columns, walls)
5  b[column|wall|space] = Deassociater(a, [1].ProvidesBoundaries)
6  c[column|wall] = NewOperator(b)
7
8  func NewOperator(a[])
9  (
10   a[] = Projector(a, [1], [2])
11 )
```

Auflistung 6.14: Beispielhaft [t]QL4BIM-Anfrage zur Demonstration des Variablenscopes

Das Ziel der Scope- und Übergaberegeln in QL4BIM ist die Unterstützung von Benutzern mit geringen Programmierkenntnissen beim Einsatz der internen und benutzerdefinierten Operatoren. So müssen alle zu verarbeitenden Variablen explizit an einen Operator als Parameter übergeben werden. An den übermittelten Daten kann ein Operator keine Veränderungen vornehmen die im übergeordneten, aufrufenden Scope sichtbar werden. Nur die erzwungene Zuordnung des Operatorergebnisses an eine Variable stellt die explizite Kommunikation zwischen Geltungsbereichen dar. Dadurch agieren Operatoren in der Anfragesprache stets lokal und ohne Seiteneffekte.

6.2.6 Argumente mit variabler Anzahl

QL4BIM-Operatoren liegen meist in mehreren Überladungen vor, die sich in ihrer Signatur unterscheiden. Aber auch eine einzige Überladung muss gegebenenfalls mit einer unterschiedlichen Anzahl an Argumenten aufgerufen werden können. Diese Art der Übergabe wird als *Variable Arguments (VarArgs)* bezeichnet.

Ein Fall, in dem dieses Konzept benötigt wird, ist der Projektionsoperator in QL4BIM. Dieser nimmt immer eine Relation als ersten Parameter entgegen. Die nachfolgenden, relationalen Parameter geben die zu übernehmenden relationalen Attribute aus der übergebenen Relation an. Ihre Anzahl kann somit zwischen 1 und $n - 1$ schwanken, wobei n die Stelligkeit der Eingangsrelation darstellt.

Für VarArgs besteht in QL4BIM eine Mindestanzahl von einem zu übergebenen Parameter. Außerdem sind derartige Argumente typisiert. Daher können mehrere Parameter mit unterschiedlicher Typisierung gleichzeitig als VarArgs spezifiziert werden. Um Parameter mit variierender Anzahl in Operatorsignaturen zu dokumentierten, werden Auslassungspunkte (engl. ellipsis) nach dem entsprechenden Parameter angefügt. Auflistung 6.15 zeigt eine beispielhafte

Signatur, in der eine variable Anzahl an relationalen Attributen genutzt wird. Mitunter bestehen weitere Einschränkungen bezüglich der Anzahl der Parameter, beispielsweise auf Grund der Stelligkeit einer relationalen Variable.

```
1 OperatorA(relation : Relation, attribute : RelationalAttribute...) : Relation
```

Auflistung 6.15: Beispielsignaturen eines Operators mit VarArgs

Die Unterstützung einer variablen Anzahl an Argumenten kann auf Grund der Semantik eines Operators zwingend erforderlich sein. Mit VarArgs kann zudem die Anzahl an benötigten Operatorüberladungen reduziert werden. Dies wirkt sich positiv auf den Implementierungsaufwand der QL4BIM-Operatoren aus.

6.2.7 Operatoren als Mittel zur Abstraktion

Im Gegensatz zu universellen Programmiersprachen ist QL4BIM speziell für die Analyse von IFC-basierten BIM-Daten auf einem hohen Abstraktionsniveau ausgelegt. Daher muss eine Abwägung zwischen der Feingranularität der Sprachoperatoren, der damit verbundenen Generalität des Ansatzes und der Beherrschbarkeit durch Endnutzer vorgenommen werden. Wird die Granularität von Operatoren maximal verfeinert, entwickelt sich die Anfragesprache zu einer universellen *Turing-vollständige*n Programmiersprache wie Java, C# oder C++ (Butterfield & Ekembe Ngondi, 2016). Die damit verbundene hohe Ausdrucksstärke kann vom Endanwender jedoch nicht vorteilhaft genutzt werden, da simultan die Komplexität der Sprache zunimmt.

Demgegenüber steht eine Sprache mit hochspezialisierten Operatoren, die bestimmte Konstellationen in Gebäudemodellen identifizieren. Hierzu sei ein imaginärer QL4BIM-Operator betrachtet, der das Enthaltensein eines bestimmten Entitätstyps in einer räumlichen Struktur untersucht, beispielsweise die Zuordnung zwischen IfcColumn- und IfcBuildingStorey-Instanzen. Derartige *High-Level*-Operatoren sind einfach anzuwenden. Fehlt jedoch für einen speziellen Anwendungsfall einer dieser Operatoren, kann die Anfragesprache nicht genutzt werden.

Um einen geeigneten Kompromiss zwischen Sprachkomplexität und universeller Einsetzbarkeit zu erreichen, werden QL4BIM-Operatoren in drei unterschiedlichen Abstufungen vorgehalten. Dabei handelt es sich um **elementare**, **zusammengesetzte (sekundäre)** und **benutzerdefinierte Operatoren**. Elementare Operatoren stellen grundlegende Funktionen der Anfragesprache bereit. Dazu zählt die Auswahl von Entitäten auf Basis von Typzugehörigkeit und Attributwerten. Sekundäre Operatoren nutzen intern elementare Operatoren und verknüpfen diese. Die Operatorengruppe dient in QL4BIM speziell für die einfache Ver-

arbeitung von komplexen IFC-Strukturen. Durch benutzerdefinierte Operatoren besteht die Möglichkeit die Sprache mit neuen Operatoren zu erweitern.

In QL4BIM werden keine Kontrollstrukturen wie Schleifen und Wenn-Dann-Abfragen für die direkte Nutzung durch den Endanwender bereitgestellt. Diese sind in Operatoren verborgen, wodurch sich der Umfang der Anfragesprache reduziert und deren Abstraktionsgrad erhöht. Durch die Kombination elementarer Operatoren ergibt sich dennoch die Möglichkeit vielfältige Analyseziele zu erreichen. Dies erhöht die Generalität der Anfragesprache, da nicht alle erdenklichen Anwendungsfälle durch sekundäre High-Level-Operatoren abgebildet werden können. Mit benutzerdefinierten Operatoren besteht zudem die Möglichkeit, Anfragen auf Basis von elementaren und sekundären Operatoren als wiederverwendbare Einheiten in QL4BIM zu integrieren.

6.2.8 Kombinierte IFC-CityGML-Analyse

CityGML und IFC stellen die zwei wichtigsten, herstellerneutralen Modellierungsansätze für Stadt- und Gebäudemodelle dar. Obwohl beide Schemata zur Repräsentation semantischer und räumlicher Daten dienen, bestehen Unterschiede in der Modellierung. Generell ist CityGML für die großflächige Repräsentation städtischer Bebauung ausgelegt. Das IFC-Schema hingegen ist für die detaillierte Beschreibung einzelner, in Planung befindlicher Bauwerke konzipiert. Dies spiegelt sich in der feingranularen semantischen, räumlichen und relationalen Modellierung und den unterstützten Geometrierepräsentationen wider.

Trotz dieser konzeptionellen Unterschiede müssen Daten der Geoinformatik und des Bauingenieurwesens zusammen ausgewertet werden. Die Unterstützung für eine derartige integrierte Auswertung ist derzeit durch die domänenspezifischen Werkzeuge beider Disziplinen nur eingeschränkt gegeben. Lösungsansätze basieren daher unter anderem auf der bidirektionalen Konvertierung (El-Mekawy et al., 2011), auf der Integration von IFC-Daten in CityGML (de Laat & van Berlo, 2011) und auf der Ableitung von LoD3-CityGML-Modellen aus IFC-Daten (Donkers, 2013).

Die von El-Mekawy et al. (2011) entwickelte bidirektionalen Konvertierung basiert auf einem vereinheitlichten Modellschema für CityGML- und IFC-Daten. Die angestrebte Konvertierung erfolgt in beide Richtungen über ein vereinheitlichtes Modell, das *Unified Building Model*. Dieses wurde als Obermenge beider ursprünglichen Modelle entwickelt. Dazu wurden alle IFC- und CityGML-Entitäten in das Modell aufgenommen und identische Konzepte zusammengeführt. Für eine detaillierte Erörterung des Unified Building Models wird auf (El-Mekawy, 2010) und (El-Mekawy et al., 2012) verwiesen.

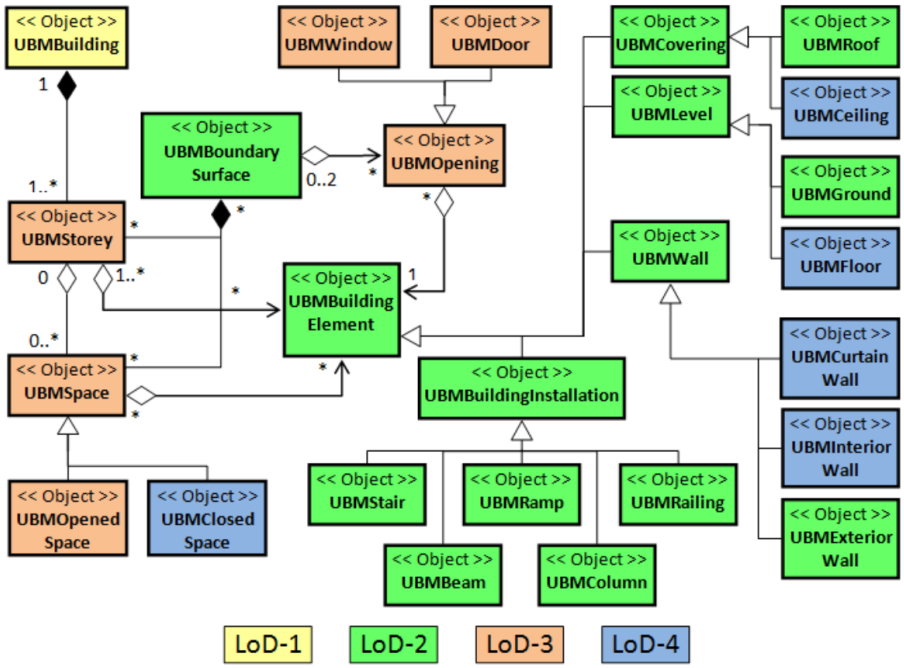


Abbildung 6.11: Das Unified Building Model, aus (El-Mekawy et al., 2011, S. 9)

Es wird davon ausgegangen, dass beide betrachteten Schemata fundierte Modellierungsstrategien darstellen, die nicht ohne Verlust an Ausdrucksstärke ineinander zu überführen sind. In QL4BIM wird daher keine Konvertierung ausgeführt, sondern IFC- und CityGML-Entitäten als generische Objekte innerhalb eines gemeinsamen Datenpools vorgehalten. Um beide Entitätsarten dennoch miteinander zu verknüpfen, dient primär der gemeinsame Raumbezug. Durch diesen und die in QL4BIM bereitgestellten räumlichen Operatoren kann somit eine integrierte Analyse erfolgen. Außerdem können attributive Filterungen und Mengenoperationen eingesetzt werden.

6.3 Die QL4BIM-Grammatik

Dieser Abschnitt beinhaltet die formale Definition der Query Language for 4D Building Information Models als kontextfreie Grammatik. Zu Beginn erfolgt die Festlegung der **t**extuellen Notation, die fortan als $^{\mathbf{t}}$QL4BIM bezeichnet wird. Danach werden Elemente der Grammatik mit visuellen Elementen verknüpft, wodurch sich die **v**isuelle Notation $^{\mathbf{v}}$QL4BIM ergibt.

Eine kontextfreie Grammatik wird von Sipser (2003, Seite 94) wie folgt definiert:

> „A context-free grammar is a 4-tuple (V, Σ, R, S), where V is a finite set called the variables, Σ is a finite set, disjoint from V, called the terminals, R is a finite

> set of rules, with each rule being a variable and a string of variables and terminals, and $S \in V$ is the start variable.“

Kontextfrei bedeutet hier, dass eine Regel stets eine **einzige** Variable auf andere Variable und Terminale abbildet. In der *Chomsky Hierarchie*, einer Klassifizierung formaler Sprachen bezüglich ihrer Komplexität, entspricht diese Art der Grammatik *Type 2* und damit der zweit niedrigsten Komplexität (Butterfield & Ekembe Ngondi, 2016).

6.3.1 Die textuelle QL4BIM-Grammatik und ihre Notation ^{t}QL4BIM

Zur Beschreibung kontextfreier Grammatiken hat sich die Metasprache *Extended Backus-Naur Form (EBNF)* etabliert (JTC, 1996). Auflistung 6.16 zeigt die EBNF-Definition von ^{t}QL4BIM. Die Ziffern in eckigen Klammern dienen zur Referenzierung der entsprechenden Produktion.

```
[01] tQL4BIM ::= statement* function*
[02] statement ::= variable '=' expression
[03] expression ::= operator '(' argument ( ',' argument)* ')'
[04] variable ::= identifier | relation
[05] operator ::= [A-Z] (a-zA-Z0-9)*
[06] identifier ::= [a-z] (a-zA-Z0-9)*
[07] externalIdentifier ::= [^(".:<>=!~)]*
[08] argument ::= simpleType | identifier | relAttributeArgument |
                  attributeAccess | predicate
[09] relation ::= identifier '[' (identifier ('|' identifier)+ )*  ']'
[10] relAttributeArgument ::= identifier? '[' (number|identifier) ']'
[11] simpleType ::= number | float | string | logic
[12] sign ::= [+-]?
[13] number ::= sign [0-9]+
[14] float ::= sign [0-9]+ '.' [0-9]+ ('E' sign [0-9]+)?
[15] string ::= '"' [^"]* '"'
[16] logic ::= 'false' | 'true' | 'unknown'
[17] predicate ::= typePredicate | attributePredicate | countPredicate
[18] attributePredicate::= attributeAccess compare
                    (simpleType | attributeAccess)
[19] attributeAccess ::= setAttributeAccess | relAttributeAccess
[20] setAttributeAccess ::= identifier '.' externalIdentifier
[21] relAttributeAccess ::= relAttributeArgument '.' externalIdentifier
[22] typePredicate ::= (identifier | relAttributeArgument) 'is'
                    externalIdentifier
[23] countPredicate ::=  relAttributeArgument compare number
[24] compare ::= '<' | '>' | '>=' | '<=' | '!=' | '=' | '~'
[25] function ::= 'func' operator(':' [A-Z][A-Z]([A-Z])? )?
                    '(' arguments ')' '(' statement+ ')'
```

Auflistung 6.16: Die ^{t}QL4BIM-Grammatik in EBNF

In diesem Kapitel wurden bereits Beispiele textueller Anfragen präsentiert. Im Folgenden wird der Bezug zwischen konkreten Anfrageelementen zu formalen Grammatikelementen verdeutlicht. Diesbezüglich ist in Auflistung 6.17 eine syntaktisch korrekte QL4BIM-Anfrage dargestellt. Da die Semantik der Anfrage nicht im Vordergrund steht, werden hier abstrakte Operatoren genutzt. Die Anfrage wird auf Basis der EBNF-Produktionen aus Auflistung 6.16 und Nennung der entsprechenden Produktion erläutert.

```
1 identifier1 = OperatorA("aString")
2 identifier2 = OperatorB(identifier1, 5.12)
3 identifier1 = MOP(identifier2)
4
5 func MyOperator:MOP(identifierA)
6 (
7   identifierB = OperatorC(identifierA, 56)
8 )
```

Auflistung 6.17: Beispielhafte [t]QL4BIM-Anfrage zur Verknüpfung mit formalen Grammatikelementen

Der erste Teil der Anfrage beinhaltet drei Statements (Produktionen 1 u. 2). Statements enthalten Ausdrücke (*expressions*), die sich aus einem Operator und Argumenten zusammensetzen (Prod. 3 u. 4). Als erster Operator wird `OperatorA` genutzt (Pro. 5). Die Argumente des ersten Ausdrucks bestehen aus einer einzelnen Zeichenkette (Prod. 8 u. 15). Im zweiten Ausdruck ist ein Bezeichner und eine Fließkommazahl in den Argumenten enthalten (Prod. 6 u. 14). Neben einem Ausdruck beinhaltet ein Statement stets eine zu anfangs genannte Variable (Prod. 4). In dieser Anfrage werden ausschließlich einfache Bezeichner eingesetzt, die für mengenbasierte Variable stehen (Prod. 6).

Ab der fünften Zeile der Anfrage beginnt die Definition eines Operators (Prod. 25). Als dessen Bezeichnung wird `MyOperator` festgelegt. Außerdem kann der alias `MOP` verwendet werden. Der benutzerdefinierte Operator wird über seine Kurzform in Zeile 3 der Anfrage aufgerufen. Die Parameterdefinition beinhaltet einen einfachen Bezeichner. Im zweiten Klammerblock des Operators ist ein einzelnes Statement enthalten (Prod. 2). Dieses nutzt `OperatorC` als Operator. Die Argumente des Statements sind der `identifierA`-Parameter und eine Integerzahl (Prod. 13).

Im nachfolgenden Syntaxbeispiel in Auflistung 6.18 kommen Relationen und relationale Attributparameter zum Einsatz. In der ersten und der letzten Aussage werden Relationen als Variable verwendet (Prod. 09). In der zweiten Aussage wird eine Relation dem Operator übergeben. Da ein Relationsbezeichner einen *identifier* darstellt, kann es als Argument genutzt werden (Prod. 08). Sollen anonyme bzw. benannte Attribute als Parameter verwendet werden, muss wie beschrieben der 1-basierte numerische Index bzw. der entsprechende Bezeichner in eckigen Klammern übergeben werden (Prod. 10). In Zeile 3 der Anfrage kommt

das relationale Attribut `ident2` zum Einsatz, das in Zeile 1 als Bestandteil von Relation `a` deklariert wurde.

```
1 a[ident1|ident2] = OperatorA(identifier)
2 identifier = OperatorB(a)
3 b[] = OperatorC(a, [ident2])
```

Auflistung 6.18: Beispielhafte tQL4BIM-Anfrage mit Relationen und relationalen Attributparametern

Das dritte Syntaxbeispiel in Auflistung 6.19 illustriert die Verwendung von AttributeAccess-Argumenten (Prod. 19). `OperatorA` in Zeile 3 wird ein mengenbasierter attribut access übergeben (Prod. 20). `OperatorB` nimmt einen relationalen attribute access entgegen (Prod. 21). Im ersten Aufruf von `OperatorB` in Zeile 4 wird das relationale Attribut als numerischer Index abgebildet. Im zweiten Aufruf von `OperatorB` wird das relationale Attribut als ident übergeben (Prod. 10).

```
1 identifier = ...
2 a[identA|identB] = ...
3 ... = OperatorA(identifier.anAttribute)
4 ... = OperatorB(a, [2].anAttribute)
5 ... = OperatorB(a, [identB].anAttribute)
```

Auflistung 6.19: Beispielhafte tQL4BIM-Anfrage mit Attributzugriffen

Auflistung 6.20 beinhaltet eine Anfrage, in der Prädikate als Parameter verwenden werden (Prod. 8 u. 17). In den ersten beiden Ausdrücken wird ein attributives Prädikat genutzt (Prod. 18). In Zeile 2 der Anfrage wird dieses aus einem AttributeAccess, einem Vergleichszeichen (Prod. 24) und einer Fließkommazahl gebildet. Das zweite attributive Prädikat nutzt zwei relationale Attributzugriffe. Im letzten Ausdruck in Zeile 4 wird ein typbasiertes Prädikat übergeben (Prod. 22).

```
1 ...
2 identifier = OperatorA(ident.attribute1 >= 22.5)
3 b[] = OperatorB(a, [1].attribute1 != [3].attribute2)
4 c[] = OperatorC(a, [2] is externalType)
```

Auflistung 6.20: Beispielhafte tQL4BIM-Anfrage mit Prädikaten

Abschließend zeigt Auflistung 6.21 vier nicht zulässige Aussagen. In der ersten Aussage wird versucht, eine mengenbasierte Variable einer weiteren zuzuweisen ohne einen Operator aufzurufen. Die reine Zuweisung von Variablen über den Assignment-Operator = ist in QL4BIM nicht möglich (Prod. 2). Dies gilt sowohl für mengenbasierte als auch für relationale Variablen. In der dritten Aussage ist keine Variable enthalten und der Assignmentoperator fehlt (Prod.

3). In Zeile 4 sind zwei Operatoren ineinander verschachtelt. Ein Operator ist als Argument jedoch nicht zulässig (Prod. 8).

```
1 identifierA = identifierB
2 a[] = b[]
3 OperatorA("aString")
4 anIdentifier = OperatorB(OperatorC("aString"), 34)
```

Auflistung 6.21: Vier syntaktisch inkorrekte Aussagen in ^{t}QL4BIM

6.3.2 Die visuelle QL4BIM-Grammatik und ihre Notation ^{v}QL4BIM

Die Vorteile einer visuellen Programmierung wurden seit dem Aufkommen dieses Ansatzes vielfach diskutiert. Schiffer (1998, Kap. 3) beschäftigt sich eingehend mit fünf häufig genannten Thesen, die für eine visuelle Programmierung sprechen. Diese lauten:

- Visuelle Programme erleichtern die Kommunikation
- Visuelle Programme sind leicht verständlich
- Visuelle Programmierung ist leicht erlernbar
- Visuelle Programmierung überwindet Sprachbarrieren
- Visuelle Programmierung heißt optimale Kognition

Durch eine detaillierte Analyse findet Schiffer (1998) für alle Thesen schlüssige Gegenargumente. Der Grund, dass in dieser Arbeit dennoch eine visuelle Sprache entwickelt wurde, liegt an den Eigenschaften von BIM-Analysen.

Im Gegensatz zu allgemeiner Programmierung sind Anfragen auf Gebäudemodelle in ihrem Umfang begrenzt und eignen sich dadurch ideal für eine visuelle Programmierung. Zudem spricht der aktuelle Erfolg von Systemen wie *Dynamo* und die damit einhergehende Nutzerakzeptanz dafür, diesen Ansatz im BIM-Bereich einzusetzen.

In der [v]QL4BIM übernehmen die eingeführten graphischen Elemente syntaktische Aufgaben, die in der ersten Variante textuell angezeigt werden müssen. Daher sind Anpassungen der ursprünglichen Grammatik nötig. Die geänderten Produktionen und Terminale sind in Auflistung 6.22 dargestellt. Die komplette Grammatik der visuellen Notation wird im Anhang in Auflistung A.5 aufgeführt.

```
1 [01] vQL4BIM ::= statement*
2 [09] relation ::= identifier identifier (identifier)+
3 [15] string ::= [^"]*
```

Auflistung 6.22: Die [v]QL4BIM-Definition in EBNF, nur Abweichungen zur [t]QL4BIM-Definition

Wie in Tabelle 6.4 dargestellt, werden Produktionen der Grammatik mit graphischen Repräsentationen, sogenannten *visuals*, verknüpft. Diese können entsprechenden der grammatikalischen Regeln mit textuellen Terminalen belegt werden. Zur Definition von Anfragen werden visuals in einer zweidimensionalen Arbeitsfläche arrangiert und miteinander in Beziehung gebracht.

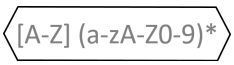
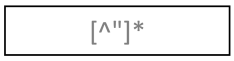
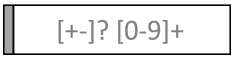
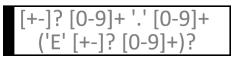
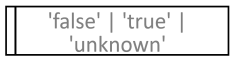
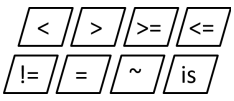
Produktion / Terminal	visuelle Darstellung
identifier	
relation	
operator	
string	
number	
float	
logical	
externalIdentifier	
compare	

Tabelle 6.4: Zuordnung von grammatikalischen Elementen zu visuellen Darstellungen

Um visuals untereinander zu verknüpfen, werden in [v]QL4BIM gerichtete Kanten eingesetzt. Operatoren verfügen daher über *Konnektoren* die als Anfangs- und Endpunkte für Verknüp-

fungen dienen. Konnektoren existieren in zwei Ausprägungen. Standardmäßig befinden sich an jedem Operator linksseitig die *Argumentkonnektoren*. Rechtsseitig ist ein *Variablenkonnektor* angeordnet. Gerichtete Kanten zeigen auf Argumentkonnektoren. Sie beginnen an Variablenkonnektoren.

Die Ausrichtung der Konnektoren kann bei Bedarf an einem Operator getauscht werden. Dann befindet sich der Variablenkonnektor linksseitig und die Argumentkonnektoren rechtsseitig. Durch diese Umkehrfunktionalität ergeben sich für Anfragen übersichtlichere Layouts. Zur Klarheit erhalten richtungsveränderte Operatoren oben links einen schwarzen Marker.

Konnektoren sind in ᵛQL4BIM je nach unterstütztem Typ unterschiedlich gestaltet. Die Repräsentationen sind in Tabelle 6.5 dargestellt. Alle Argumentkonnektoren können als VarArgs-Varianten auftreten. In diesem Fall wird nach Belegung eines VarArgs-Konnektors ein weiterer Konnektor angedeutet wenn dessen Verwendung im konkreten Fall möglich ist.

Konnektor für	visuelle Darstellung	Konnektor für	visuelle Darstellung
Set		Relation	
RelationalAttribute		String	
Number		Float	
Logical		AttributeAccess	
Predicate		VarArg	

Tabelle 6.5: Die typisierten Konnektoren in ᵗQL4BIM

Zur Demonstration der sich ergebenden visuellen Notation, wird Musteranfrage 4 und die textuellen Beispiele aus Auflistungen 6.17-6.21 in ᵛQL4BIM überführt. Abbildung 6.12 zeigt Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand).

In der visuellen Umsetzung von Musteranfrage 4 belegt jeweils eine Aussage den horizontalen Bereich der Arbeitsfläche. Die Verknüpfung von Variablen erfolgt durch eine vom Variablenkonnektor ausgehende gerichtete Kante zwischen Operator und Variable. Dies stellt die Variablenzuweisung in ᵛQL4BIM dar. Mengenbasierte Variablen werden als abgerundete Rechtecke und relationale Variable als abgefaste Rechtecke dargestellt. Variablen werden durch Kanten an die Operatoren übergeben. Außerdem kommen externe Bezeichner, die als Rechteck mit Raute visualisiert werden, als Parameter vor. Zur übersichtlicheren Gestaltung der Anfrage wurde die Operatorausrichtung der beiden Typfilter getauscht.

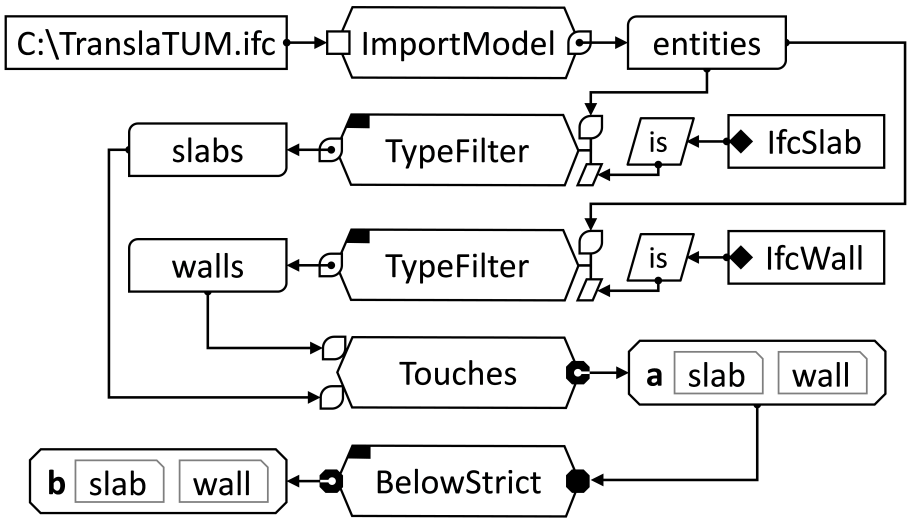


Abbildung 6.12: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ᵛQL4BIM

In Abbildung 6.13 wurde die Anfrage aus Auflistung 6.17 in ᵛQL4BIM umgesetzt. In der visuellen Notation wird die Definition von benutzerdefinierten Operatoren nicht unterstützt. Die Anfrage besteht daher aus drei Statements. Als Parameter kommen mengenbasierte Variablen und eine Fließkommazahl zum Einsatz.

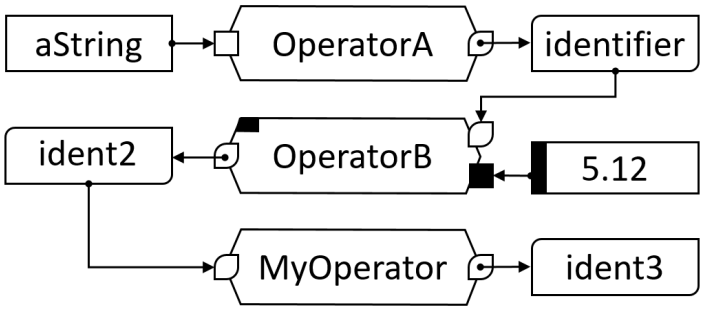


Abbildung 6.13: Visuelle Notation der textuellen Anfrage aus Auflistung 6.17

Das nachfolgende Beispiel in Abbildung 6.14 entspricht Auflistung 6.18. Hier wird der Einsatz von relationalen Variablen in ᵛQL4BIM veranschaulicht. Das Ergebnis von `OperatorA` wird einer relationalen Variablen zugewiesen. `OperatorB` erhält diese Relation als Parameter. Im nachfolgenden Statement wird `OperatorC` neben der Relation zusätzlich ein relationales Attribut übergeben.

Das dritte Beispiel in Abbildung 6.15 verdeutlicht den AttributeAccess in der visuellen Notation der Anfragesprache. Der Punkt-Operator der textuellen Notation wird durch die Verknüpfung von Konnektoren ersetzt. Im ersten Statement wird dem Operator eine Mengenvariable übergeben. In den nachfolgenden Fällen wird eine relationale Variable verwendet. Hier muss

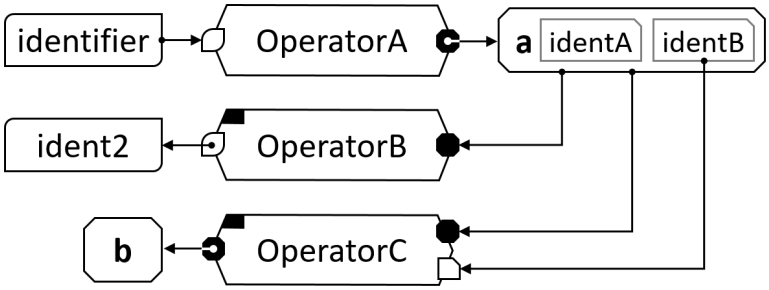


Abbildung 6.14: Visuelle Notation der textuellen Anfrage aus Auflistung 6.18

zusätzlich der relationale Index spezifiziert werden. Dies kann über eine gerichtete Kante mit numerischem Index geschehen, der von der Relation zum Konnektor führt. Die zweite Möglichkeit besteht im Andocken der Kante direkt am relationalen Attribut. Die Variante mit numerischem Index ist primär für Relationen mit anonymen Attributen gedacht. Die Spezifizierung des Entitätsattributs geschieht in allen drei Statements über den ExternalIdentifier-Konnektor.

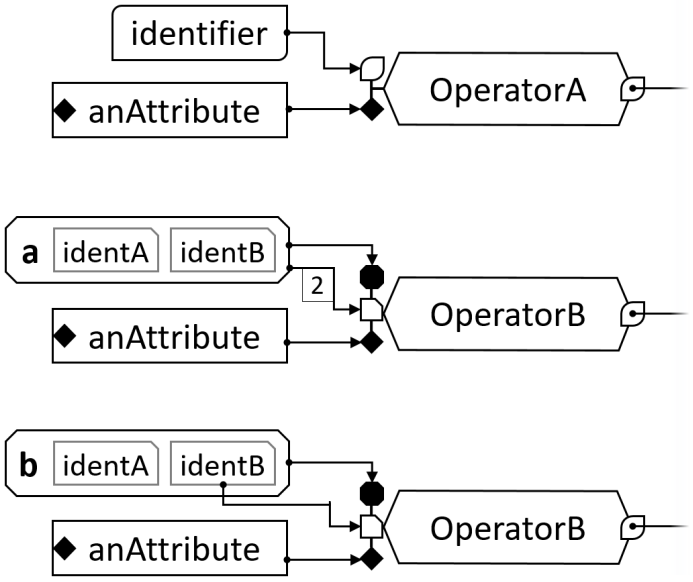


Abbildung 6.15: Visuelle Notation der textuellen Anfrage aus Auflistung 6.19

Visuelle Aussagen unter Verwendung von Prädikaten sind in Abbildung 6.16 aufgezeigt. `OperatorA` wird ein Prädikat aus Entitätsattribut und einer Fließkommazahl zugeordnet. Innerhalb der zweiten Operation werden zwei Entitätsattribute im Prädikat verglichen. Im letzten Statement erfolgt eine Typprüfung durch das Prädikat.

Dem textuellen Beispiel entsprechend zeigt Abbildung 6.17 vier nicht zulässige Aussagen. In den ersten beiden Aussagen wird eine mengenbasierte bzw. eine relationale Variable direkt einer weiteren zugeordnet. Im dritten Statement wird dem Variablenkonnektor von `OperatorA`

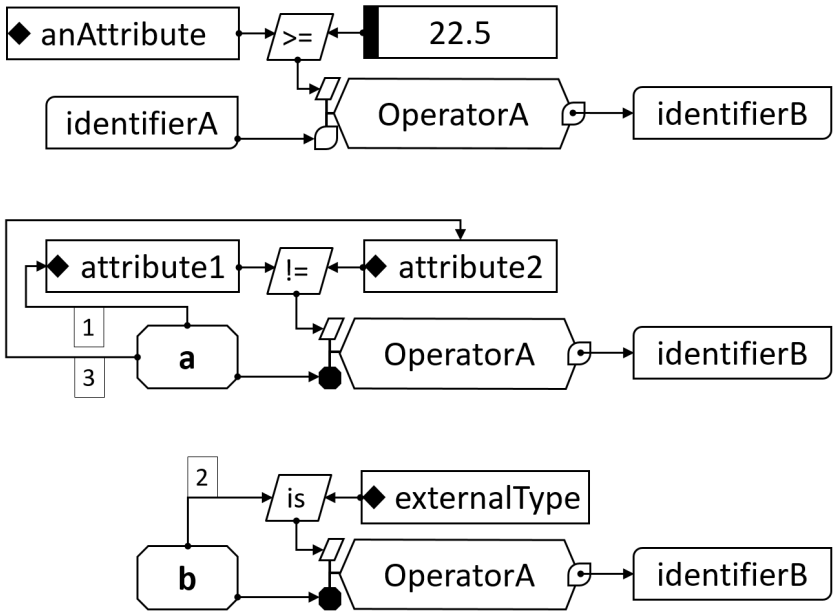


Abbildung 6.16: Visuelle Notation der textuellen Anfrage aus Auflistung 6.20

keine Variable zugeordnet. Danach wird `OperatorB` direkt `OperatorC` als Argument zugewiesen. Jedes dieser vier Statements ist in ᵛQL4BIM nicht zulässig.

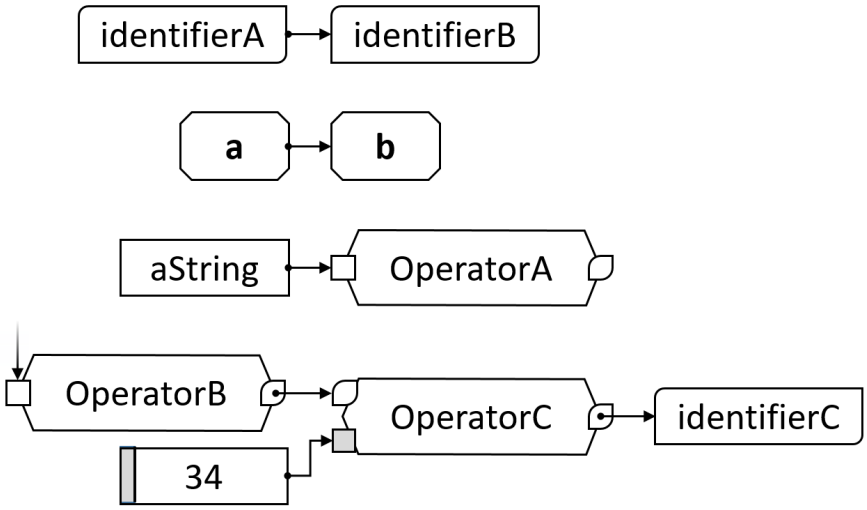


Abbildung 6.17: Vier syntaktisch inkorrekte Aussagen in ᵛQL4BIM

Die Nutzung von typisierten Konnektoren an Operatoren unterstützt den Nutzer in der Erstellung valider Anfragen. Gleichzeitig wird die visuelle Notation dadurch komplexer und unter Umständen unübersichtlich. Daher wird festgelegt, dass das Laufzeitsystem der Anfragesprache die Möglichkeit bieten muss Konnektoren auszublenden. Wenn möglich erfolgt außerdem die Vereinigung von eingehenden Kanten an Operatoren. Zur Verdeutlichung zeigt Abbildung 6.18 Musteranfrage 4 in der vereinfachten Ansicht ohne typisierte Konnektoren.

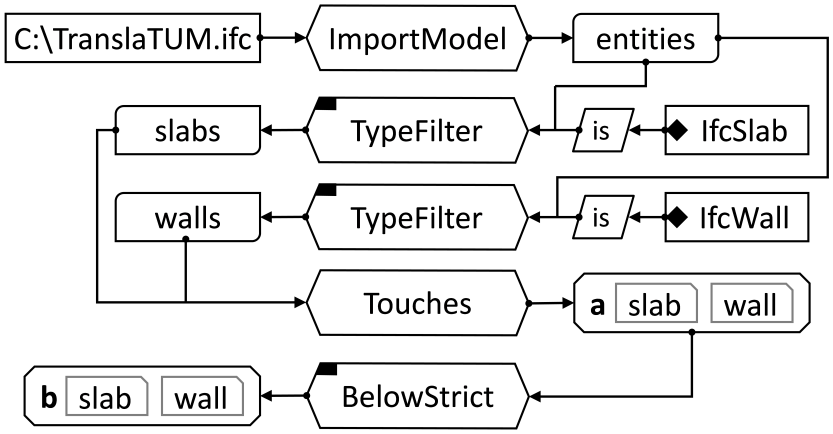


Abbildung 6.18: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ᵛQL4BIM, vereinfachte Ansicht ohne typisierte Konnektoren

Die reduzierte Grammatik der textuellen Notation konnte auch in ᵛQL4BIM angewendet werden. Konzeptionell orientiert sich der visuelle Ansatz an den imperativen Eigenschaften einer steuerflussbasierten Umgebung, ohne jedoch Kontrollstrukturen nach außen zu tragen. Derartige Strukturen werden hingegen, wie in ᵗQL4BIM, in unterschiedlichen Varianten durch Operatorüberladungen verwirklicht. Die Ausführungsreihenfolge von Ausdrücken wird im visuellen Ansatz durch die sequenzielle Verknüpfung von Variablen, Konstanten und Operatoren bestimmt. Zusätzlich wird die Anordnungsreihenfolge dieser Elemente in der graphischen Benutzeroberfläche berücksichtigt. Vorteil der visuellen Notation von ᵛQL4BIM ist die klare Typisierung von Sprachkonstrukten und entsprechenden Operatorkonnektoren. Damit wird dem Endanwender verdeutlicht welche Elemente miteinander verknüpft werden können. Durch die Deaktivierung der Konnektorvisualisierung besteht gleichzeitig die Möglichkeit vereinfachte, visuelle Anfragen zu erhalten. In diesen sind die einzelnen Anweisungen klar nachvollziehbar. Es wird erhofft, dass über den visuellen Ansatz auch Endanwender ohne Programmierkenntnisse die Sprache intuitiv nutzen können.

6.4 Überblick über die QL4BIM-Operatoren

Wie bereits erwähnt, gliedern sich die Operatoren der Anfragesprache in die drei Stufen elementarer, sekundärer und benutzerdefinierter Operatoren. Die nachfolgenden Tabellen geben einen Überblick über die Operatoren der unterschiedlichen Stufen. Tabelle 6.6 zeigt alle elementaren, nicht-räumlichen Operatoren. Die elementaren, räumlichen Operatoren sind in Ta-

belle 6.7 enthalten. Tabelle 6.8 führt die sekundären Operatoren der Anfragesprache auf und Tabelle 6.9 enthält die später diskutierten, benutzerdefinierten Operatoren.

Im nachfolgenden Teil dieses Kapitels werden die Operatoren der drei Bereiche detailliert vorgestellt. Dies umfasst die Semantik des jeweiligen Operators und die verfügbaren Überladungen. Beispielhafte textuelle Anfragen verdeutlichen die Entwicklungen. Zu Gunsten der Übersichtlichkeit werden die Operatoraliase nur bei den benutzerdefinierten Operatoren aufgeführt. Auf eine visuelle Umsetzung der Beispielanfragen wird aus Platzgründen verzichtet. Die visuelle Notation kann jedoch mit dem in Kapitel 7 entwickelten Laufzeitsystem automatisch aus der textuellen Notation abgeleitet werden.

Gruppe	Operator	Alias	Beschreibung
Import / Export	ImportModel	IM	Import einer IFC- oder CityGML Instanzdatei
	ExportModel	EM	Export einer IFC- oder CityGML Instanzdatei
Relationale Algebra	TypeFilter	TF	Selektion nach Typzugehörigkeit
	AttributeFilter	AF	Selektion über Attributwert
	JoinResolver	JR	Verknüpfung auf Basis identischer Entitäten
	Projector	PJ	Projektion relationaler Variablen
	Dereferencer	DR	Auflösung attributiver Referenzbeziehungen
	LinkCounter	LC	Selektion über die Anzahl an Beziehungen
Boolesche Operationen	Union	UN	Vereinigung von Mengen oder Relationen
	Difference	DIF	Differenz von Mengen oder Relationen
Extremwert-operationen	Minimum	MIN	Selektion nach minimalen Attributwert
	Maximim	MAX	Selektion nach maximalen Attributwert

Tabelle 6.6: Elementare, nicht räumliche Operatoren von QL4BIM

Gruppe	Operator	Alias	Beschreibung
Topologisch	Disjoint	DI	Topologisch Disjunkt
	Contains	CT	Topologisch Enthalten
	Inside	IN	Topologisch Innerhalb
	Equals	EQ	Topologische Gleichheit
	Covers	CV	Topologische Bedeckung
	CoveredBy	CB	Topologisch Abgedeckt
	Overlaps	OL	Topologische Überschneidung
	Touches	TO	Topologische Berührung
Metrisch	Nearer	NE	Näher als Grenzwert
	Farther	FU	Weiter als Grenzwert
Strikt direktional	AboveStrict	AS	Striktes Darüber
	BelowStrict	BS	Striktes Darunter
	NorthStrict	NS	Striktes Nördlich
	SouthStrict	SS	Striktes Südlich
	WestStrict	WS	Striktes Westlich
	EastStrict	ES	Striktes Östlich
Relaxiert direktional	AboveRelaxed	AR	Relaxiertes Darüber
	BelowRelaxed	BR	Relaxiertes Darunter
	NorthRelaxed	NR	Relaxiertes Nördlich
	SouthRelaxed	SR	Relaxiertes Südlich
	WestRelaxed	WR	Relaxiertes Westlich
	EastRelaxed	ER	Relaxiertes Östlich

Tabelle 6.7: Elementare, räumliche Operatoren von QL4BIM

6.5 Elementare Operatoren

Die erste Gruppe beinhaltet elementare Operatoren zur Analyse extern definierter Entitäten. Derartige Operatoren stellen die feingliedrigsten Analysefunktionen in QL4BIM bereit, die zudem nicht durch andere Operatoren nachgebildet werden können.

Neben Import- und Exportfunktionalität werden Operationen aus der relationalen Algebra wie Projektion, Theta-Selektion und der natürlicher Join eingeführt. Außerdem erlauben elementare Operatoren die Auflösung von Verweise zwischen Entitäten und die Ausführung von Typanfragen. Domänenspezifische Operatoren zur Umsetzung räumlicher Untersuchungen komplementieren diese Stufe.

Gruppe	Operator	Alias	Beschreibung
Dereferenzierung	Deassociater	DA	Dereferenzierung inverser Beziehungen
	TimeResolver	TR	Aufgabenbezogene Dereferenzierung
	PropertyFilter	PF	Eigenschaftsmenge
Zeitlich	Before	BF	Zeitliches Davor
	After	AF	Zeitliches Danach
	Meets	ME	Zeitliches Zusammentreffen
	Encloses	EC	Zeitliches Enthaltensein
	During	DU	Zeitliches Während
	Starts	ST	Gleichzeitiger Beginn
	Ends	EN	Gleichzeitiges Ende
	Overlays	OL	Zeitliche Überlagerung
	Identical	ID	Zeitliche Gleichheit
Extraktion u. Reintegration	MvdFilter	MF	Teilmodellextraktion per MVD
	Merger	ME	Three-way-merger mit Geometrieauswertung

Tabelle 6.8: Sekundäre Operatoren von QL4BIM

Operator	Alias	Beschreibung
WallsInBasement	WIB	Selektion von Wänden, die sich im Kellergeschoss befinden
NextTaskWithLag	NTL	Dereferenzierung nachfolgender Aufgaben mit Pufferkriterium

Tabelle 6.9: Beispielhafte benutzerdefinierte Operatoren in QL4BIM

6.5.1 Input- und Output-Funktionalität

Um Instanzdateien in die Ablaufumgebung der Anfragesprache zu importieren und gefilterte Modelle als eigenständige Dateien abzulegen, stellt QL4BIM den `ImportModel`- und den `ExportModel`-Operator bereit.

ImportModel-Operator

Semantik: Der `ImportModel`-Operator dient zum Import von IFC- und CityGML-Daten in das Anfragesystem. Dazu wird der Pfad zu einer entsprechenden Instanzdatei als Argument übergeben. Der Operator liefert eine Menge an IFC-Entitäten bzw. komplexer CityGML-Objekte zurück.
Überladung 1: Die zu ladende Instanzdatei wird über das String-typisierte `path`-Argument übergeben. Zur Speicherung der Entitäten muss eine mengentypisierte Variable verwendet werden. Die Signatur des Operators ist textuell in Auflistung 6.23 und graphisch in Ab-

bildung 6.19 dargestellt. Auflistung 6.24 zeigt ein Beispiel zur Nutzung des `ImportModel`-Operators.

```
ImportModel(path : String) : Set
```

Auflistung 6.23: [t]QL4BIM-Signatur des ImportModel-Operators

Abbildung 6.19: [v]QL4BIM-Repräsentation des ImportModel-Operators

```
entities = ImportModel("C:\TranslaTUM.ifc")
```

Auflistung 6.24: Beispiel zur Nutzung des ImportModel-Operators in [t]QL4BIM

ExportModel-Operator

Semantik: Der `ExportModel`-Operator dient zum Export von IFC- und CityGML-Daten. Dazu wird dem Operator eine Menge an Entitäten und ein Dateipfad übergeben. Je nach genutzter Dateiendung werden nur IFC- bzw. CityGML-Entitäten gespeichert. Das Ergebnis der Operation umfasst die mitunter aufbereitete Menge an exportierten Entitäten. So ist für einen validen IFC-Export die Erstellung einer Hierarchie räumlicher Strukturen zwingend notwendig. Daher werden in diesem Fall neben Eigenschaftsmengen auch assoziierte, räumliche Strukturen in die finale Instanzdatei mit aufgenommen.
Überladung 1: Über den `entities`-Parameter wird die Menge zu exportierender Entitäten übergeben. Das String-typisierte `path`-Argument gibt Speicherort und Namen der zu schreibenden Datei an. Die Signatur des Operators ist textuell in Auflistung 6.25 und graphisch in Abbildung 6.20 dargestellt.

```
ExportModel(entities : Set, path : String) : Set
```

Auflistung 6.25: [t]QL4BIM-Signatur des ExportModel-Operators

Abbildung 6.20: [v]QL4BIM-Repräsentation des ExportModel-Operators.

Auflistung 6.26 zeigt ein Beispiel zur Nutzung des `ExportModel`-Operators. Nach dem Import eine Entitätsmenge wird diese durch eine Typfilterung auf IfcWall-Instanzen reduziert. Die neu erhaltene Menge wird der Variable `walls` zugewiesen. In Zeile 3 erfolgt der Export der Menge durch den `ExportModel`-Operator in die Datei `walls.ifc`.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 walls = TypeFilter(entities is IfcWall)
3 exportedEntities = ExportModel(walls, "C:\walls.ifc")
```

Auflistung 6.26: Beispiel zur Nutzung des ExportModel-Operators in tQL4BIM.

6.5.2 Funktionalitäten der relationalen Algebra

Die Schemata der Bau- und Geoinformatik definieren Vererbungshierarchien, die zur Filterung von Modellen eingesetzt werden können. Zudem wird eine Selektion von Entitäten auf Basis von Attributwerten benötigt. Diese Funktionalitäten werden durch Theta-Selektionen innerhalb des `TypeFilter`- und des `AttributeFilter`-Operators realisiert. Weitere auf der relationalen Algebra basierende Analysemöglichkeiten umfassen den natürlichen Verbund (`RelationResolver`-Operator), den Verbund auf Basis von Entitätsreferenzen (`Dereferencer`-Operator) und die Projektion (`Projector`-Operator). Der `LinkCount`-Operator filtert Relationen auf Basis ihrer Kardinalität.

TypeFilter-Operator

Semantik: Der `TypeFilter`-Operator selektiert aus einer übergebenen Menge oder Relation Entitäten, die Instanzen einer angegebenen Klasse darstellen. Dabei wird die Klassenhierarchie des genutzten Entitätsschemas beachtet. So werden neben Instanzen der genannten Klasse auch Instanzen von Subklassen in das Ergebnis eingeschlossen.
Überladung 1: In dieser Überladung wird die Datenquelle über eine im Typprädikat enthaltene Variable festgelegt. Das Prädikat beinhaltet außerdem den `is`-Vergleichsoperator und eine Typspezifikation. Der zu selektierende Typ wird über ein ExternalType-Argument festgelegt. Als Ergebnis wird eine Menge zurückgegeben. Die Signatur des Operators ist textuell in Auflistung 6.27 und graphisch in Abbildung 6.21 dargestellt.

```
TypeFilter(predicate : SetTypePredicate) : Set
```

Auflistung 6.27: tQL4BIM-Signatur des TypeFilter-Operators, Überladung 1

Abbildung 6.21: vQL4BIM-Repräsentation des TypeFilter-Operators, Überladung 1

Auflistung 6.28 zeigt ein Beispiel zur Nutzung des Operators. In der ersten Zeile von Auflistung 6.28 enthält die Variable `entities` unterschiedliche EXPRESS-Typen. Als Ergebnis des

TypeFilters beinhaltet Variable `walls` in Zeile 2 nur IfcWall-Instanzen und entsprechende Subtypen, hier IfcWallStandardCase.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 walls = TypeFilter(entities is IfcWall)
```

Auflistung 6.28: Beispiel zur Nutzung des TypeFilter-Operators in ᵗQL4BIM, Überladungen 1

Überladung 2: In dieser Überladung stellt die `relation`-Variable die auszuwertende Datenquelle in Form einer Relation dar. Die relevanten Relationsattribute werden über das Typprädikat-Argument festgelegt. Da es sich hierbei um ein Argument mit variabler Anzahl handelt, können mehrere Typprädikate übergeben werden. Somit können mehrere relationale Attribute durch einen Operatoraufruf auf Typzugehörigkeit untersucht werden. Als Ergebnis wird eine Relation zurückgegeben, deren Stelligkeit mit der der Eingangsrelation identisch ist. In dieser sind ausschließlich Tupel enthalten in denen die Typprüfung für alle auszuwertenden relationalen Attribute `true` zurückliefert. Die Signatur des Operators ist textuell in Auflistung 6.29 und graphisch in Abbildung 6.22 dargestellt.

```
TypeFilter(relation : Relation, predicates : TypePredicate...) : Relation
```

Auflistung 6.29: ᵗQL4BIM-Signatur des TypeFilter-Operators, Überladung 2

Abbildung 6.22: ᵛQL4BIM-Repräsentation des TypeFilter-Operators, Überladung 2

Auflistung 6.30 zeigt ein Beispiel zur Nutzung der zweiten Überladung. Nach dem Entitätsimport erfolgt die referenzielle Auflösung des *Decomposes*-Attributs. Hierdurch erhält man eine zweistellige Relation mit unterschiedlichen Typen in jedem relationalen Attribut. Der `TypeFilter` ermöglicht es, die Relation in unterschiedlichen Attributen zu filtern (Z. 3). So wird jedes Tupel im Attribut `whole` auf den Typ IfcRoot geprüft. Falls dies der Fall ist, wird der zweite Index des Tupels (Attribut `part`) auf den Typ IfcSlab hin untersucht. Als Ergebnis sind in der Variable `a[roof|slab]` in Zeile 3 nur IfcWall-Instanzen und entsprechende Subtypen im ersten Tupelindex vorhanden. Im zweiten Index sind ausschließlich IfcSlab und deren Subtypen enthalten.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[whole|part] = Deassociater(entities.Decomposes)
3 a[roof|slab] = TypeFilter(a, [whole] is IfcRoof, [part] is IfcSlab)
```

Auflistung 6.30: Beispiel zur Nutzung des TypeFilter-Operators in ᵗQL4BIM, Überladungen 2

AttributeFilter-Operator

Semantik: Der `AttributeFilter`-Operator selektiert aus einer übergebenen Menge oder Relation Entitäten, deren Attribute Prädikaten genügen. Damit realisiert der Operator die Theta-Selektion der relationalen Algebra. Zum Beispiel kann somit geprüft werden, ob ein Attributwert einer Entität größer einer benutzerdefinierten Schranke ist oder eine bestimmte Zeichenkette enthält.
Überladung 1: In dieser Überladung wird die Datenquelle über eine im Attributprädikat inkludierte Set-Variable festgelegt. Dieses Prädikat enthält neben einem attribute access einen Vergleichsoperator und eine einfache Konstante. Somit wird die zu untersuchende Datenquelle und das auszuwertende Entitätsattribut spezifiziert. Als Ergebnis wird eine Menge zurückgegeben. Die Signatur der Überladung ist textuell in Auflistung 6.31 und graphisch in Abbildung 6.23 dargestellt.

```
AttributeFilter(predicate : SetAttributePredicate) : Set
```

Auflistung 6.31: [t]QL4BIM-Signatur des AttributeFilter-Operators, Überladung 1

Abbildung 6.23: [v]QL4BIM-Repräsentation des AttributeFilter-Operators, Überladung 1

Auflistung 6.32 zeigt ein Beispiel zur Nutzung der ersten Überladung. Nach dem Entitätsimport erfolgt eine typbasierte Filterung. Damit sind in der Variable `doors` in Zeile 2 ausschließlich IfcDoor und davon abgeleitete Typen enthalten. Der `AttributeFilter` nimmt die mengenbasierte Variable `doors` als Teil des Attributprädikats entgegen (Z. 3). Mit Hilfe des Attributzugriffs wird das *OverallHeight*-Attribut ausgewählt und mit einer numerischen Konstante verglichen. Somit sind in der mengenbasierten Variable `highDoors` ausschließlich IfcDoor-typisierte Instanzen enthalten, deren *OverallHeight*-Attribut größer oder gleich `2.4` Einheiten beträgt.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 doors = TypeFilter(entities is IfcDoor)
3 highDoors = AttributeFilter(doors.OverallHeight >= 2.4)
```

Auflistung 6.32: Beispiel zur Nutzung des AttributeFilter-Operators in [t]QL4BIM, Überladungen 1

Überladung 2: In dieser Überladung stellt die *relation*-Variable die auszuwertende Datenquelle in Form einer Relation dar. Die auszuwertenden Relationsattribute werden über das Attributprädikat-Argument festgelegt. Da es sich hierbei um ein Argument mit variabler Anzahl handelt, können mehre Attributprädikate übergeben werden. Somit können mehre-

re relationale Attribute und damit Entitäten eines Tupels auf bestimmte Attributwerte hin untersucht werden. Als Ergebnis wird eine Relation zurückgegeben, deren Stelligkeit mit der Eingangsrelation identisch ist. In dieser sind ausschließlich Tupel enthalten, in denen die Attributwerte für alle auszuwertenden relationalen Attribute den übergebenen Prädikaten genügen. In dieser Überladung erfolgt der Attributzugriff durch Verknüpfung eines relationalen Attributs mit dem Punktoperator und einem externen Attributbezeichner. Der erste Teil des Zugriffs spezifiziert damit den Tupelindex, der zweite das auszuwertende Entitätsattribut. Zudem besteht die Möglichkeit zwei Entitätsattribute eines Tupels miteinander zu vergleichen. Die Signatur der Überladung ist textuell in Auflistung 6.33 und graphisch in Abbildung 6.24 dargestellt.

```
AttributeFilter(relation : Relation, predicates : AttributePredicate...)
                : Relation
```

Auflistung 6.33: [t]QL4BIM-Signatur des AttributeFilter-Operators, Überladung 2

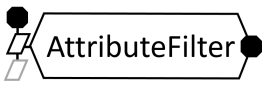


Abbildung 6.24: [v]QL4BIM-Repräsentation des AttributeFilter-Operators, Überladung 2

Auflistung 6.34 zeigt ein Beispiel zur Nutzung der zweiten Überladung. Am Anfang wird eine zweistellige Relation aus sich berührenden IfcWall- und IfcDoor-Instanzen gebildet (Z. 1 - 4). Somit sind in der Variable `a` im ersten Index Wände und im zweiten Türen vorgehalten. Nun werden drei unterschiedliche Attributfilterungen durchgeführt. In der ersten wird das `wall`-Attribut der Relation ausgewählt und das *OverallWidth*-Attribut der enthaltenen Entitäten mit einer Schranke verglichen (Z. 5). In Zeile 6 werden die Entitäten eines Tupels untereinander verglichen. Hier erfolgt der Attributzugriff über die numerischen Indizes. Im vorherigen Operatoraufruf wurde hingegen der relationale Attributbezeichner genutzt.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 walls = TypeFilter(entities is IfcWall)
3 doors = TypeFilter(entities is IfcDoor)
4 a[wall|door] = Touches(walls, doors)
5 b[wall|door] = AttributeFilter(a, [wall].OverallWidth >= 2400)
6 c[wall|door] = AttributeFilter(a, [1].OverallWidth < [2].OverallWidth)
```

Auflistung 6.34: Beispiel zur Nutzung des AttributeFilter-Operators in [t]QL4BIM, Überladungen 2

JoinResolver-Operator

Semantik: Der `JoinResolver`-Operator verbindet zwei relationale Variablen auf Basis identischer Entitäten. Das verbindende, relationale Attribut wird nur einmal in das Ergebnis aufgenommen. Dadurch ergibt sich die Stelligkeit der zurückgegebenen Relation als eins geringer als die Summe der Stelligkeiten der Eingangsrelationen. Für die Reihenfolge von Attributen in der Ergebnisrelation gilt folgende Regel. Die Attribute von Relation `relation1` werden in ihrer Reihenfolge unverändert übernommen. Danach folgen die Attribute von Relation `relation2`, mit Ausnahme des Verbindungsattributs. In die Ergebnisrelation werden nur Tupel mit Übereinstimmungen aufgenommen. Somit realisiert der *JoinResolver* einen inneren Equi-Join (siehe Kapitel 4, Abbildung 4.5).
Überladung 1: Die Argumente `relation1` und `relation2` geben die zu verarbeitenden Datenquellen an. Die relationalen Attribute werden durch die Argumente `relAttribute1` und `relAttribute2` spezifiziert. Die Signatur des Operators ist textuell in Auflistung 6.35 und graphisch in Abbildung 6.25 dargestellt.

```
JoinResolver(relation1 : Relation, relAttribute1 : RelAtt,
             relation2 : Relation, relAttribute2 : RelAtt) : Relation
```

Auflistung 6.35: [t]QL4BIM-Signaturen des JoinResolver-Operators

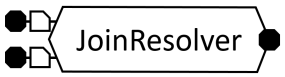


Abbildung 6.25: [v]QL4BIM-Repräsentation des JoinResolver-Operators

Auflistung 6.36 zeigt ein Beispiel zur Nutzung des `JoinResolvers`. Im ersten Teil der Anfrage werden IfcWall- und IfcWindow-typisierte Entitätsmengen erstellt (Z. 1-3). Durch die Dereferenzierung inverser Attribute erhält man Wände mit ihren Durchbrüchen in Form von Wand-Öffnungspaaren (Z. 4). Ebenso können Fenster mit deren beherbergten Durchbrüchen als Fenster-Öffnungspaare selektiert werden (Z. 5). Durch den `JoinResolver` werden in Zeile 6 Wand-Öffnungspaare und Fenster-Öffnungspaare verbunden, wenn die Öffnung in beiden Paaren identisch ist. Als Ergebnis erhält man eine dreistellige Relation aus Wand-Öffnung-Fenster-Tripeln. Das Beispiel zeigt außerdem, dass bei der Übergabe eines relationalen Attributs der Relationsbezeichner wiederholt werden kann, siehe Argumente `a[opening]` und `b[opening]` in Zeile 6.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 walls = TypeFilter(entities is IfcWall)
3 windows = TypeFilter(entities is IfcWindow)
4 a[wall|opening] = Deassociater(walls.HasOpenings)
5 b[window|opening] = Deassociater(windows.FillsVoids)
6 c[wall|opening|window] = JoinResolver(a, a[opening], b, b[opening])
```

Auflistung 6.36: Beispiel zur Nutzung des JoinResolver-Operators in tQL4BIM, Überladungen 1

Projector-Operator

Semantik: Der `Projector`-Operator übernimmt ausschließlich diejenigen Attribute einer Relation, die als Argumente übergeben werden. Dadurch können relationale Attribute aus Relationen entfernt werden, falls diese in nachfolgenden Untersuchungen nicht benötigt werden.
Überladung 1: In dieser Überladung wird eine relationale Variable als Datenquelle übergeben. Danach folgt ein einzelnes Argument vom Typ RelationalAttribute. Da somit ein einzelnes relationales Attribut ausgewählt wurde, liefert die Überladung eine mengenbasierte Variable zurück. Die Signatur der Überladung ist textuell in Auflistung 6.37 und graphisch in Abbildung 6.26 dargestellt.

```
Projector(relation : Relation, attribute : RelAtt) : Set
```

Auflistung 6.37: tQL4BIM-Signatur des Projector-Operators, Überladung 1

Abbildung 6.26: vQL4BIM-Repräsentation des Projector-Operators, Überladung 1

Auflistung 6.38 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport wird in Zeile 2 eine Relation aus Entität-OwnerHistory-Paaren erstellt. In Zeile 3 wird die Relation `a` dem `Projector` übergeben. Danach folgt die Spezifizierung eines einzelnen relationalen Attributs, hier über den numerischen Index. Somit sind in der Variable `ownerHistories` alle IfcOwnerHistory-Instanzen des Modells ohne Duplikate vorgehalten.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity, ownerHistory] = Dereferencer(entities.OwnerHistory)
3 ownerHistories = Projector(a, [2])
```

Auflistung 6.38: Beispiel zur Nutzung des Projector-Operators in tQL4BIM, Überladungen 1

Überladung 2: Auch in dieser Überladung wird eine einzelne relationale Variable als Datenquelle übergeben. Danach folgen jedoch mindestens zwei Argumente vom Typ Relatio-

nalAttribute. Somit liefert die Überladung eine relationale Variable zurück. Die Stelligkeit der Ergebnisrelation ist identisch mit der Anzahl an RelationalAttribute-Argumenten. Die Signatur der Überladung ist textuell in Auflistung 6.39 und graphisch in Abbildung 6.27 dargestellt.

```
Projector(relation : Relation, attribute : RelAtt,
          moreAtts : RelAtt...) : Relation
```

Auflistung 6.39: [t]QL4BIM-Signatur des Projector-Operators, Überladung 1

Abbildung 6.27: [v]QL4BIM-Repräsentation des Projector-Operators, Überladung 1

Auflistung 6.40 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport und einer Typfilterung wird durch den `Deassocitater`-Operator eine Relation aus Produkten und ihren Materiallayern gewonnen. Durch mehrfaches Dereferenzieren kann von diesen Layern bis zum eigentlichen Material navigiert werden. Man erhält somit in Zeile 5 eine vierstellige Relation aus Produkt, Layer, Materiallayer und Material. Um letztendlich für alle Produkte der Instanzdatei das assoziierte Material in einer zweistelligen Relation vorzuhalten, wird der `Projektor` eingesetzt (Z. 6). Diesem wird die vierstellige Relation `a` und die relationalen Attribute `product` und `material` übergeben. Als Endergebnis erhält man eine zweistellige Relation aus Produkt-Material-Paaren. Das Beispiel zeigt zudem, wie durch den negativen Index `-1` stets auf letzte, neu hinzugekommene relationale Attribute zugegriffen werden kann (Z. 4 u. 5).

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 products = TypeFilter(entities is IfcProduct)
3 a[product|layerset] = Deassociater(products.HasAssociations)
4 b[product|layerset|layer] = Dereferencer(a, [-1].MaterialLayer)
5 c[product|layerset|layer|material] = Dereferencer(b, [-1].Material)
6 d[product|material] = Projector(c, [product], [material])
```

Auflistung 6.40: Beispiel zur Nutzung des Projector-Operators in [t]QL4BIM, Überladungen 2

Dereferencer-Operator

Semantik: Der `Dereferencer`-Operator löst attributive Referenzbeziehungen zwischen Entitäten auf. Die Dereferenzierung basiert auf der Nennung von Entitätsattributen in einem SetAttributeAccess. Damit eine Dereferenzierung erfolgen kann, müssen folgende Bedingungen erfüllt sein. Erstens muss das genannte Attribut in der untersuchten Entität vorhanden sein. Das Attribut beinhaltet zudem keinen einfachen Datentyp, sondern verweist auf eine oder

mehrere weitere Entitäten. Aus der ursprüngliche Entität wird mit der dereferenzierten Entität ein neues Tupel erstellt. Im Falle von containbasierten Attributen werden diese somit in 1:1-Beziehungen umgewandelt.

Überladung 1: In den ersten Überladungen wird ein mengenbasierter Attributzugriff als Argument übergeben. Dieser besteht aus einer Set-Variable, dem Punkt-Operator und einem externen Bezeichner für das zu dereferenzierende Entitätsattribut. Als Ergebnis liefert die Überladung eine zweistellige Relation zurück. Die Signatur der Überladung ist textuell in Auflistung 6.41 und graphisch in Abbildung 6.28 dargestellt.

```
Dereferencer(attributeAccess : SetAttributeAccess) : Relation
```

Auflistung 6.41: [t]QL4BIM-Signatur des Dereferencer-Operators, Überladung 1

Abbildung 6.28: [v]QL4BIM-Repräsentationen des Dereferencer-Operators, Überladung 1

Auflistung 6.42 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Import der Entitäten wird in Zeile 2 der `Dereferencer`-Operator aufgerufen. Als Datenquelle dient die mengenbasierte Variable `entities`. Durch den Attributzugriff wird die *OwnerHistory*-Eigenschaft jeder enthaltenen Entität dereferenziert. Dies ist möglich, da diese Eigenschaft auf eine eigenständige Entität verweist, hier vom Typ IfcOwnerHistory. Die Anfrage liefert somit eine zweistellige Relation aus Entität-OwnerHistory-Paaren.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|ownerHistory] = Dereferencer(entities.OwnerHistory)
```

Auflistung 6.42: Beispiel zur Nutzung des Dereferencer-Operators in [t]QL4BIM, Überladungen 1

Überladung 2: In dieser Überladung wird eine Relation und ein relationaler Attributzugriff übergeben. Dieser besteht aus einem relationalen Attribut, dem Punkt-Operator und einem externen Bezeichner für das zu dereferenzierende Entitätsattribut. Als Ergebnis liefert die Überladung eine Relation zurück. Deren Stelligkeit ist um eins höher als die Stelligkeit der Eingangsrelation. Das dereferenzierte Attribut wird in die Ergebnisrelation als letztes Attribut angefügt. Die Signatur der Überladung ist textuell in Auflistung 6.43 und graphisch in Abbildung 6.29 dargestellt.

```
Dereferencer(relation: Relation, attributeAccess : RelAttributeAccess)
            : Relation
```

Auflistung 6.43: [t]QL4BIM-Signatur des Dereferencer-Operators, Überladung 2

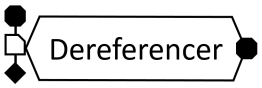


Abbildung 6.29: [v]QL4BIM-Repräsentationen des Dereferencer-Operators, Überladung 2

Auflistung 6.44 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird durch den `Deassocitater`-Operator eine Relation aus Produkten und ihren assoziierten Entitäten gewonnen. Ein weiterer `TypeFilter` reduziert die Selektion auf Produkt-Aufgaben-Paare (Z. 4). Im `Dereferencer`-Operator in Zeile 5 wird das zweite Attribut der Eingangsrelation untersucht. Das zu dereferenzierende Entitätsattribut lautet *TaskTime*. Darin hält jede IfcTask zeitbezogene Daten vor. Somit liefert die Anfrage eine dreistellige Relation aus Produkt, Aufgabe und Aufgabenzeit.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 products = TypeFilter(entities is IfcProduct)
3 a[product|entity] = Deassociater(products.HasAssociations)
4 b[product|task] = TypeFilter(a, [2] is IfcTask)
5 c[product|task|tasktime] = Dereferencer(b, [2].TaskTime)
```

Auflistung 6.44: Beispiel zur Nutzung des Dereferencer-Operators in [t]QL4BIM, Überladungen 2

LinkCounter-Operator

Semantik: Der Operator filtert eine Relation auf Basis der Kardinalität eines relationalen Attributs. Somit kann die Anzahl an Beziehungen (engl. links), in denen eine Entität involviert ist, als Selektionskriterium verwendet werden.
Überladung 1: In dieser Überladung stellt die `relation`-Variable die zu verwendende Datenquelle dar. Das auszuwertende Relationsattribut wird über das CounterPredicate-Argument festgelegt. Dieses setzt sich aus einem relationalen Attribut, einem Vergleichsoperator und einer Integer-Konstanten zusammen. Die Stelligkeit der Ergebnisrelation ist identisch mit der der Eingangsrelation. Die Signatur des Operators ist textuell in Auflistung 6.45 und graphisch in Abbildung 6.30 dargestellt.

```
RelationCounter(relation : Relation, predicate : CounterPredicate) : Relation
```

Auflistung 6.45: [t]QL4BIM-Signatur des LinkCounter-Operators, Überladung 1

Abbildung 6.30: [v]QL4BIM-Repräsentation des LinkCounter-Operators, Überladung 1

Auflistung 6.46 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Import der Entitäten erfolgt die referenzielle Auflösung des *IsGroupedBy*-Attributs. Die erhaltene zweistellige Relation wird auf IfcDistributionSystem- und IfcPipeSegment-Instanzen gefiltert (Z. 3). Der `LinkCounter` selektiert in Zeile 4 diejenigen Tupel, in denen das identische Verteilungssystem mindestens fünfmal genutzt wird. Als Ergebnis der Anfrage erhält man somit eine Relation aus Verteilungssystem-Rohr-Paaren, wobei das System aus mindestens fünf Rohren besteht.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[group, part] = Deassociater(entities.IsGroupedBy)
3 b[system, pipe] = TypeFilter(a, [1] is IfcDistributionSystem,
                                  [2] is IfcPipeSegment)
4 c[system, pipe] = LinkCounter(b, [system] >= 5)
```

Auflistung 6.46: Beispiel zur Nutzung des LinkCounter-Operators in ᵗQL4BIM

6.5.3 Boolesche Operatoren

Im Folgenden werden die Codd'schen, booleschen Operatoren in ihren QL4BIM-Varianten erläutert. Diese umfassen den `Union`- und den `Difference`-Operator. Durch diese Operatoren können Prädikate indirekt verknüpft und negiert werden. So müssen Prädikate nicht in einem einzigen Operator erweitert werden, was dem Sprachparadigma einfacher Statements widerspräche. Ein `Intersection`-Operator ist in der Anfragesprache nicht nötig, da diese Funktionalität durch wiederholte, sequenzielle Filterung umgesetzt werden kann. Der `Union`-Operator ermöglicht hingegen eine Oder-Verknüpfung von Prädikaten, der `Difference`-Operator die Negation eines Prädikats.

Ursprünglich wurden die booleschen Operatoren für relationale Datenbanken entwickelt, in denen ausschließlich atomare Werte in Tupeln zulässig sind. Diese ursprünglichen Codd'schen Operatoren können auf Relationen angewendet werden, wenn deren Attribute zueinander kompatibel sind. Codd (1990, S. 79) formuliert dazu folgende Bedingungen.

> „Suppose that S and T are two relations. Then, S and T are unioncompatible if they are of the same degree and it is possible to establish at least one mapping between the columns of S and those of T that is one-to-one, and with the property that, for every column A of S and every column B of T, if column A is mapped onto column B, then A and B draw their values from a common domain."

Relationale und mengenbasierte Variablen in QL4BIM bestehen aus komplexen Entitäten externer Schemata. Diese Entitäten werden über den Typ ExternalEntity in der Anfragesprache abgebildet. Für die booleschen Operatoren bedeutet dies, dass stets eine gemeinsame Typdomäne zwischen zwei Relationsattributen besteht. Relationale Variable müssen somit lediglich

identische Stelligkeiten aufweisen, damit eine Verarbeitung möglich ist. Mengenbasierte Variable können immer als Eingangsquellen genutzt werden.

Union-Operator

Semantik: Der `Union`-Operator vereinigt die übergebenen mengenbasierten bzw. relationalen Variablen.
Überladung 1: In dieser Überladung stellt die `set`-Variable eine der zu verwendende Datenquelle dar. Weitere Quellen werden über das variable Argument `sets` angegeben. Somit werden mindestens zwei oder mehr mengenbasierte Variable vereinigt. Die Ergebnismenge beinhaltet keine Duplikate. Die Signatur der Überladung ist textuell in Auflistung 6.47 und graphisch in Abbildung 6.31 dargestellt.

```
Union(set : Set, sets: Set...) : Set
```

Auflistung 6.47: [t]QL4BIM-Signatur des Union-Operators, Überladung 1

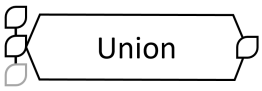


Abbildung 6.31: [v]QL4BIM-Repräsentation des Union-Operators, Überladung 1

Auflistung 6.48 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung werden durch die folgenden `AttributeFilter` drei Mengen an unterschiedlichen Rohrarten gewonnen (Z. 3 - 5). Um diese Mengen zu vereinigen wird in Zeile 6 der `Union`-Operator aufgerufen. Die Entitäten der unterschiedlichen Rohrarten werden somit in der Variable `certainPipes` vorgehalten.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 pipes = TypeFilter(entities is IfcPipeSegment)
3 culverts = AttributeFilter(pipes.PredefinedType = CULVERT)
4 gutters = AttributeFilter(pipes.PredefinedType = GUTTER)
5 spools = AttributeFilter(pipes.PredefinedType = SPOOL)
6 certainPipes = Union(culverts, gutters, spools)
```

Auflistung 6.48: Beispiel zur Nutzung des Union-Operators in [t]QL4BIM, Überladung 1

Überladung 2: In dieser Überladung stellt die `relation`-Variable eine der zu verwendende Datenquelle dar. Weitere Quellen werden über das variable Argument `relations` angegeben. Somit werden mindestens zwei oder mehr Relationen vereinigt. Dazu müssen alle Eingangsrelationen in ihrer Stelligkeit identisch sein. Die Ergebnisrelation beinhaltet keine identischen Tupel und weist die Stelligkeit der Eingangsrelationen auf. Die Signatur der Überladung ist textuell in Auflistung 6.49 und graphisch in Abbildung 6.32 dargestellt.

```
Union(relation : Relation, relations : Relation...) : Relation
```

Auflistung 6.49: [t]QL4BIM-Signatur des Union-Operators, Überladung 2

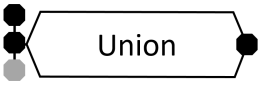


Abbildung 6.32: [v]QL4BIM-Repräsentation des Union-Operators, Überladung 2

Auflistung 6.50 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird der `Deassociator`-Operator aufgerufen. Somit werden alle Rohre, die mit einer Steuereinheit assoziiert sind, in einer zweistelligen Relation vorgehalten (Z. 3). Durch die folgenden `AttributeFilter` werden zwei Relationen aus Rohr-Steuerungselement-Paaren gebildet, wobei sich die Rohrarten jeweils unterscheiden (Z. 4 u. 5). Durch Aufruf des `Union`-Operators in Zeile 6 werden die Tupel beider Relationen vereint und in der relationalen Variable `d[somePipe|control]` gespeichert.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 pipes = TypeFilter(entities is IfcPipeSegment)
3 a[pipe|control] = Deassociator(pipes.HasControlElements)
4 b[culvert|control] = AttributeFilter(a, [1].PredefinedType = CULVERT)
5 c[gutter|control] = AttributeFilter(a, [1].PredefinedType = GUTTER)
6 d[somePipe|control]  = Union(b, c)
```

Auflistung 6.50: Beispiel zur Nutzung des Union-Operators in [t]QL4BIM, Überladung 2

Difference-Operator

Semantik: Der `Difference`-Operator bildet die Differenzmenge aus den zwei übergebenen mengenbasierten bzw. relationalen Variablen.
Überladung 1: Das Ergebnis der Überladung ist die Menge `setA` minus der Vereinigungsmenge aller `setBs` Argumente. Die Signatur der Überladung ist textuell in Auflistung 6.51 und graphisch in Abbildung 6.33 dargestellt.

```
Difference(setA : Set, setBs: Set...) : Set
```

Auflistung 6.51: [t]QL4BIM-Signatur des Difference-Operators, Überladung 1

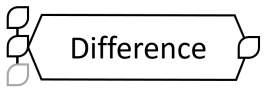


Abbildung 6.33: [v]QL4BIM-Repräsentation des Difference-Operators, Überladung 1

Auflistung 6.52 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und typbasierter Filterung werden in der `buildingElements`-Variable ausschließlich IfcBuildingElements vorgehalten. Durch den nachfolgenden Typfilter wird aus dieser Mengen der Subtyp IfcCovering selektiert und der Variable `coverings` zugewiesen (Z. 3). Das Vorgehen wird in Zeile 4 für IfcShadingDevice, ebenfalls Subtyp von IfcBuildingElement, wiederholt. In Zeile 5 werden von allen IfcBuildingElements diejenigen entfernt, die dem Subtyp IfcCovering oder IfcShadingDevice aufweisen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 buildingElements = TypeFilter(entities is IfcBuildingElement)
3 coverings = TypeFilter(buildingElements is IfcCovering)
4 shadings = TypeFilter(buildingElements is IfcShadingDevice)
5 certainElements = Difference(buildingElements, coverings, shading)
```

Auflistung 6.52: Beispiel zur Nutzung des Difference-Operators in [t]QL4BIM, Überladung 1

Überladung 2: Das Ergebnis der Überladung ist die Relation `relationA` minus der Vereinigungsrelation aller `relationBs` Argumente. Die Signatur der Überladung ist textuell in Auflistung 6.53 und graphisch in Abbildung 6.34 dargestellt.

```
Difference(relationA : Relation, relationBs: Relation...) : Set
```

Auflistung 6.53: [t]QL4BIM-Signatur des Difference-Operators, Überladung 2

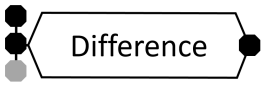


Abbildung 6.34: [v]QL4BIM-Repräsentation des Difference-Operators, Überladung 2

Auflistung 6.54 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport erfolgt eine spezielle Form der Attribut-Dereferenzierung in Zeile 2. Somit erhält man eine zweistellige Relation mit Entität-Struktur-Paaren. Durch den folgenden Typfilter werden aus dieser Relation nur diejenigen Tupel übernommen, deren Entität am Index 2 den Typ IfcBuildingStorey aufweist. In Zeile 3 wird die gewonnene Relation `b` zusätzlich gefiltert. Dies erfolgt auf Basis des Attributs *Name* der IfcBuildingStorey-Entität des Tupels. Mit dem `Difference`-Operator werden aus `b` alle Tupel entfernt die auch in `c` enthalten sind. Somit enthält die Variable `d` Entität-Stockwerk-Paare, deren Stockwerk im *Name*-Attribut nicht `basement` enthält.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|struct] = Deassociater(entities.ContainedInSpatialStructure)
3 b[entity|storey] = TypeFilter(a, [2] is IfcBuildingStorey)
4 c[entity|certainStorey] = AttributeFilter(b, [2].Name ~ "basement")
5 d[entity|certainStorey] = Difference(b, c)
```

Auflistung 6.54: Beispiel zur Nutzung des Difference-Operators in ᵗQL4BIM, Überladung 2

6.5.4 Extremwertbasierte Filterung

Extreme Attributwerte stellen einen verbreiteten Ansatz zur Filterung von Daten dar. Bei der Analyse eines 4D-Gebäudemodells können unter anderem extreme räumliche und zeitliche Ausprägungen interessant sein. Beispiel dazu sind Bauteile mit maximaler Höhe innerhalb eines Stockwerks oder die zuletzt anstehenden Aufgaben innerhalb eines Prozesses des Bauablaufplans. Die Anfragesprache stellt daher den `Minimum`- und den `Maximum`-Operator zur Filterung von Mengen- und Relationsvariablen bereit. Die Extremwert-Filterung basiert auf der Nennung eines Entitätsattributs in einem AttributeAccess. Damit eine Filterung erfolgen kann, muss das Attribut einen einfachen, sortierbaren Datentyp aufweisen.

Minimum-Operator

Semantik: Der `Minimum`-Operator ermittelt den Minimalwert eines Entitätsattributs in einer mengenbasierten oder relationalen Variable. Danach werden alle Entitäten bzw. Tupel selektiert, die diesen Minimalwert aufweisen. Der `Maximum`-Operator verhält sich identisch, bezieht sich jedoch auf den Maximalwert eines Entitätsattributs.

Überladung 1: In dieser Überladung wird ein mengenbasierter Attributzugriff als Argument übergeben. Der Zugriff besteht aus einer Set-Variable, dem Punkt-Operator und einem externen Bezeichner für das zu dereferenzierende Entitätsattribut. Als Ergebnis liefert die Überladung eine Menge zurück, in der die Entitäten mit dem niedrigsten Attributwert der Eingangsmenge enthalten sind. Die Signatur der Überladung ist textuell in Auflistung 6.55 und graphisch in Abbildung 6.35 dargestellt.

```
Minimum(attributeAccess : SetAttributeAccess) : Set
```

Auflistung 6.55: ᵗQL4BIM-Signaturen des Minimum-Operators, Überladung 1

Auflistung 6.56 zeigt ein Beispiel zur Nutzung der Überladung. Nach initialem Import und nachfolgender Typfilterung wird in Zeile 3 der `Minimum`-Operator aufgerufen. Dieser ermittelt den Minimalwert des *OverallWidth*-Attributs der übergebenen Entitäten. Danach wer-

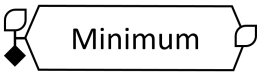


Abbildung 6.35: [v]QL4BIM-Repräsentation des Minimum-Operators, Überladung 1

den alle Entitäten selektiert, die diesen Minimalwert aufweisen. Somit liegen in der Variable `minimalWideDoors` alle IfcDoors mit dem geringstem Wert im *OverallWidth*-Attribut vor.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 doors = TypeFilter(entities is IfcDoor)
3 minimalWideDoors = Minimum(doors.OverallWidth)
```

Auflistung 6.56: Beispielhafte Aufrufe des Minimum-Operators in [t]QL4BIM, Überladung 1

Überladung 2: In dieser Überladung wird eine Relation und ein relationaler Attributzugriff übergeben. Der Attributzugriff besteht aus einem relationalen Attribut, dem Punkt-Operator und einem externen Bezeichner für das auszuwertende Entitätsattribut. Als Ergebnis liefert die Überladung eine Relation zurück, deren Stelligkeit identisch mit der Stelligkeit der Eingangsrelation ist. In der Ergebnisrelation sind die Tupel mit dem niedrigsten Attributwert der Eingangsrelation enthalten. Die Signatur der Überladung ist textuell in Auflistung 6.57 und graphisch in Abbildung 6.36 dargestellt.

```
Minimum(relation : Relation, attributeAccess : RelAttributeAccess) : Relation
```

Auflistung 6.57: [t]QL4BIM-Signatur des Minimum-Operators, Überladung 2

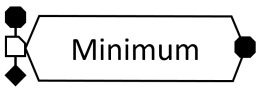


Abbildung 6.36: [v]QL4BIM-Repräsentation des Minimum-Operators, Überladung 2

Auflistung 6.58 zeigt ein Beispiel zur Nutzung der Überladung. Zu Beginn werden alle Entitäten einer Instanzdatei importiert und nachfolgend auf den Typ IfcCovering reduziert. In Zeile 3 erfolgt eine spezielle Form der Attribut-Dereferenzierung. Somit erhält man eine zweistellige Relation mit Belag-Raum-Paaren. Der nachfolgende `Mimimum`-Operator ermittelt den Minimalwert des *ElevationWithFlooring*-Attributs im zweiten Tupelindex. Danach werden alle Tupel selektiert, die diesen Minimalwert aufweisen. Somit liegen in der Variable `b[covering|lowestSpace]` alle Belag-Raum-Paare vor, in denen der Raum den geringsten Wert im *ElevationWithFlooring*-Attribut aufweist.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 coverings = TypeFilter(entities is IfcCovering)
3 a[covering|space] = Deaccociator(coverings.CoversSpaces)
4 b[covering|lowestSpace] = Minimum(a, [space].ElevationWithFlooring)
```

Auflistung 6.58: Beispiel zur Nutzung des Minimum-Operators in [t]QL4BIM, Überladung 2

Maximum-Operator

Auf Grund der Ähnlichkeit zum `Minimum`-Operator werden in Auflistung 6.59 nur die Signaturen des `Maximum`-Filters aufgeführt. In Abbildung 6.37 sind die graphischen Repräsentationen des Operators dargestellt.

```
1 Maximum(attributeAccess : SetAttributeAccess) : Set
2 Maximum(relation : Relation, attributeAccess : RelAttributeAccess) : Relation
```

Auflistung 6.59: [t]QL4BIM-Signaturen des Maximum-Operators, Überladung 1 und 2

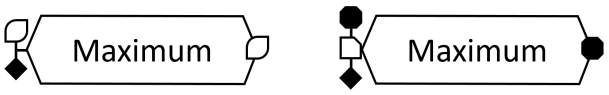


Abbildung 6.37: [v]QL4BIM-Repräsentationen des Maximum-Operators, Überladung 1 und 2

6.5.5 Räumliche Operatoren

Von Borrmann (2007, Kap.7.4) wurden drei Kategorien an räumlichen Operatoren für eine BIM-Anfragesprache vorgeschlagen und entsprechende mathematische Definitionen entwickelt. Die Kategorisierung teilt sich in topologische, direktionale und metrische Operatoren auf. Da sich durch diese Operatoren eine Vielzahl an ingenieurtechnischen Fragestellungen auf hohem Abstraktionsniveau beantworten lassen, übernimmt QL4BIM diese Einteilung und stellt die beschriebenen Operatoren bereit. Im Falle der topologischen und direktionalen Operatoren wurden außerdem erweiterte Varianten konzipiert, die benutzerdefinierte Toleranzen unterstützen. Die benötigen Algorithmen zur Umsetzung der räumlichen Operatoren wurden in Kapitel 5.2 entwickelt.

Alle räumlichen Operatoren in QL4BIM basieren auf der Verarbeitung der Boundary Representation von Entitäten. Durch das polymorphe Verhalten von Entitäten können die Signaturen dieser Operatorengruppe jedoch genauso aufgebaut werden wie die Signaturen nichträumlicher Operatoren. In der Laufzeitumgebung der Sprache wird jede Entität zur räumlichen Prozessierung in ein ISpatialRep gecastet und kann so als räumliches Objekt verarbeitet werden.

Die räumlichen Operatoren nutzen außerdem das in Abschnitt 6.2.3 vorgestellte Konzept zur Abstraktion von Kontrollstrukturen. Daher liegen alle räumlichen Operatoren in den nun beschriebenen Überladungen vor. Tabelle 6.10 zeigt die entsprechenden Signaturen und die jeweilige Kardinalität der Auswertung.

Überladung	Kardinalität
OperatorA(set1 : Set, set2 : Set) : Relation	n:m
OperatorA(relation1 : Relation, attribut1 : RelAtt, relation2 : Relation, attribute2 : RelAtt) : Relation	n:m
OperatorA(relation1 : Relation, attribute1 : RelAtt, attribute2 : RelAtt) : Relation	1:1
OperatorA(relation1 : Relation) : Relation	1:1

Tabelle 6.10: Überladungen der räumlichen Operatoren und deren Auswertungskardinalitäten

Die topologischen Operatoren Disjoint, Contains, Inside, Equal, Covers, Covered, Overlaps und Touches

Die topologischen Operatoren der Anfragesprache basieren auf dem in Abschnitt 2.4.2 diskutierten 9-Intersection Model (9-IM). In diesem werden die kombinatorischen Schnittmengen aus Innerem, Rand und Äußerem zweier räumlicher Objekte gebildet. Die sich ergebenden Ergebnismatrizen werden topologischen Prädikaten zugeordnet. In QL4BIM werden die Operatoren `Disjoint`, `Contains`, `Inside`, `Equal`, `Covers`, `Covered`, `Overlaps` und `Touches` unterstützt. Wegen der Ähnlichkeit in der Anwendung wird diese Gruppe ausschließlich am Beispiel des `Touches`-Operators erläutert.

Semantik: Der `Touches`-Operator identifiziert sich berührende Entitäten.
Überladung 1: Die Argumente `set1` und `set2` geben die zu verarbeitenden Datenquellen an. Alle Entitäten der ersten Menge werden mit allen Entitäten der zweiten Menge auf Berührung hin überprüft. Dazu werden die räumlichen Repräsentationen der Entitäten genutzt. Die Signatur der Überladung ist textuell in Auflistung 6.60 und graphisch in Abbildung 6.38 dargestellt.

```
Touches(set1 : Set, set2 : Set) : Relation
```

Auflistung 6.60: [t]QL4BIM-Signatur des Touches-Operators, Überladung 1

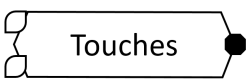


Abbildung 6.38: [v]QL4BIM-Repräsentation des Touches-Operators, Überladung 1

Auflistung 6.61 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und zweifacher Typfilterungen werden `piles` und `columns` dem `Touches`-Operator übergeben (Z. 4). Als Ergebnis liefert die Anfrage die zweistellige Relation `a`, die aus sich berührenden Fundament-Stützen-Paaren besteht.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 piles = TypeFilter(entities is IfcPile)
3 columns = TypeFilter(entities is IfcColumn)
4 a[pile|column] = Touches(piles, columns)
```

Auflistung 6.61: Beispiel zur Nutzung des Touches-Operators in ᵗQL4BIM, Überladung 1

Überladung 2: Die Argumente `relation1` und `relation2` geben die zu verarbeitenden relationalen Datenquellen an. Alle Tupel der ersten Relation werden mit allen Tupeln der zweiten Relation verglichen. Zur Auswahl, welche Entitäten aus den Tupeln jeweils auf gegenseitige Berührung hin untersucht werden, dienen die zwei relationalen Argumente `attribute1` und `attribute2`. Die Stelligkeit der Ergebnisrelation ist die Summe aus den Stelligkeiten der beiden Eingangsrelationen. Die Attribute von Relation `relation1` werden als erstes in die Ergebnisrelation übernommen. Danach folgen die Attribute von Relation `relation2`. Die Signatur der Überladung ist textuell in Auflistung 6.62 und graphisch in Abbildung 6.39 dargestellt.

```
Touches(relation1 : Relation, attribute1 : RelAtt,
        relation2 : Relation, attribute2 : RelAtt) : Relation
```

Auflistung 6.62: ᵗQL4BIM-Signatur des Touches-Operators, Überladung 2

Abbildung 6.39: ᵛQL4BIM-Repräsentation des Touches-Operators, Überladung 2

Auflistung 6.63 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport wird mit dem `Deassociator`-Operator eine spezielle Form der Attribut-Dereferenzierung angewendet (Z. 2). Somit erhält man eine zweistellige Relation. Durch den folgenden Typfilter werden nur die Tupel übernommen, deren Entität am Index `1` eine IfcPile-Instanz und am Index `2` eine IfcTask-Instanz darstellen (Z. 3). Diese typbasierte Filterung einer Relation wird für die Typen IfcColumn und IfcTask wiederholt (Z.4). In Zeile 5 wird der `Touches`-Operator mit den Relationen `b` und `c` aufgerufen. Für die erste Relation wird das relationale Attribut `pile` und für die zweite Relation `column` ausgewählt. Der Operator liefert eine vierstellige Relation aus Fundament, fundamentbezogener Aufgabe, Stütze und stützenbezogener Aufgabe. Fundament und Stütze berühren sich zudem. Das Beispiel zeigt, dass der Bezeichner der betroffenen Relation in der relationalen Attributauswahl wiederholt werden kann.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|object] = Deassociator(entities.ReferencedBy)
3 b[pile|task] = TypeFilter(a, [1] is IfcPile, [2] is IfcTask)
4 c[column|task] = TypeFilter(a, [1] is IfcColumn, [2] is IfcTask)
5 d[pile|taskP|column|taskC] = Touches(b, b[pile], c, c[column])
```

Auflistung 6.63: Beispiel zur Nutzung des Touches-Operators in tQL4BIM, Überladung 2

Überladung 3: Das Argument `relation` gibt die zu verarbeitenden relationalen Datenquellen an. Aus allen Tupel der Relation werden zwei Entitäten ausgewählt und untersucht. Zur Spezifikation, welche Entitäten aus dem Tupeln auf gegenseitige Berührung hin untersucht werden sollen, dienen die zwei relationalen Argumente `attribute1` und `attribute2`. Die Stelligkeit der Ergebnisrelation ist mit der Stelligkeit der Eingangsrelation identisch. Die Signatur der Überladung ist textuell in Auflistung 6.64 und graphisch in Abbildung 6.40 dargestellt.

```
Touches(relation : Relation, attribute1 : RelAtt,
        attribute2 : RelAtt) : Relation
```

Auflistung 6.64: tQL4BIM-Signatur des Touches-Operators, Überladung 3

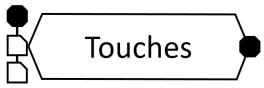


Abbildung 6.40: vQL4BIM-Repräsentation des Touches-Operators, Überladung 3

Auflistung 6.65 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird in Zeile 3 der `Deassociator`-Operator aufgerufen. Somit erhält man eine zweistellige Relation. Diese wird durch eine weitere Attribut-Dereferenzierung mit dem `Deassociator`-Operator zu einer dreistelligen Relation erweitert (Z. 4). Jedes enthaltene Tripel besteht aus Öffnung, beherbergtem Bauteil und füllendem Bauteil. Der `Touches`-Operator in Zeile 5 selektiert die Tripel, in denen sich beherbergtes und füllendes Bauteil berühren.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 openings = TypeFilter(entities is IfcOpening)
3 a[opening|host] = Deassociator(openings.VoidsElements)
4 b[opening|host|filling] = Deassociator(a, [opening].HasFillings)
5 c[opening|host|filling] = Touches(b, [host], [filling])
```

Auflistung 6.65: Beispiel zur Nutzung des Touches-Operators in tQL4BIM, Überladung 3

Überladung 4: Diese Überladung verhält sich wie Überladung 3 und untersucht stets das erste und zweite relationale Attribut der übergebenen Relation. Die Signatur der Überladung ist textuell in Auflistung 6.66 und graphisch in Abbildung 6.41 dargestellt.

```
Touches(relation : Relation) : Relation
```

Auflistung 6.66: [t]QL4BIM-Signatur des Touches-Operators, Überladung 4

Abbildung 6.41: [v]QL4BIM-Repräsentation des Touches-Operators, Überladung 4

Auflistung 6.67 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird der `Deassociator`-Operator mit `openings.VoidsElements` aufgerufen. Somit erhält man eine zweistellige Relation aus Öffnungen und deren beherbergte Bauteile. Der `Touches`-Operator in Zeile 4 selektiert alle Tupel, in denen sich Öffnung und Bauteil berühren.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 openings = TypeFilter(entities, IfcOpening)
3 a[opening|host] = Deassociator(openings.VoidsElements)
4 b[opening|host] = Touches(a)
```

Auflistung 6.67: Beispiel zur Nutzung des Touches-Operators in [t]QL4BIM, Überladung 4

Topologische Operatoren mit Toleranzunterstützung

Neben den standardmäßigen topologischen Operatoren wurden für QL4BIM Erweiterungen mit Toleranzunterstützung entwickelt. Die dafür benötigten Algorithmen wurden im vorherigen Kapitel vorgestellt.

Semantik: Die Operatoren basieren auf der Erstellung veränderter geometrischer Repräsentationen und unterstützen somit einen numerischen Toleranz-Parameter (siehe Kapitel **??**). Dadurch können Ungenauigkeiten in der geometrischen Modellierung und domänenspezifische Besonderheiten berücksichtigt werden. Die für den `tol`-Parameter genutzte Einheit muss in der Ablaufumgebung festgelegt oder aus der referenzierten Instanzdatei abgeleitet werden.
Überladungen: Siehe standardmäßige topologische Operatoren

Überladung 5-8:Die sich ergebenden zusätzlichen Überladungen sind am Beispiel des `Touches`-Operators in ihrer textuellen Form in Auflistung 6.68 dargestellt. Abbildung 6.42 zeigt die entsprechenden graphischen Repräsentationen. Die weiteren topologischen Operatoren weisen die identischen Signaturen für ihre toleranzunterstützenden Varianten auf.

```
1 Touches(set1 : Set, set2 : Set, tol : Float) : Relation
2 Touches(relation1 : Relation, attribute1 : RelAtt,
          relation2 : Relation, attribute2 : RelAtt, tol : Float) : Relation
3 Touches(relation : Relation, attribute1 : RelAtt, attribute2 : RelAtt,
          tol : Float) : Relation
4 Touches(relation : Relation, tol : Float) : Relation
```

Auflistung 6.68: [t]QL4BIM-Signaturen des Touches-Operators mit Toleranzunterstützung

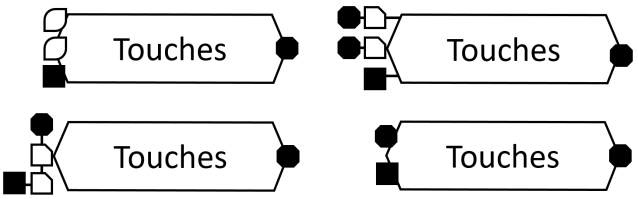


Abbildung 6.42: [v]QL4BIM-Repräsentationen des Touches-Operators, Überladung 5-8

Auflistung 6.69 zeigt ein Beispiel zur Nutzung der `Touches`-Operator mit Toleranzunterstützung. In der ersten Zeile werden alle Entitäten einer Instanzdatei importiert. Danach erfolgen zwei Typfilterungen (Z. 2 u. 3). Die so erhaltenen Platten und Stützen werden in Zeile 4 auf Berührung getestet, wobei eine Toleranz von `0.1` Einheiten berücksichtigt wird.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities, IfcSlab)
3 columns = TypeFilter(entities, IfcColums)
4 a[slab|column] = Touches(slabs, columns, 0.1)
```

Auflistung 6.69: Beispiel zur Nutzung des Touches-Operators in [t]QL4BIM, Überladung 5

Die metrischen Operatoren Nearer und Farther

Die metrischen Operatoren `Nearer` und `Farther` berechnen Minimalabstände zwischen den räumlichen Repräsentationen von Entitäten. Als Ergebnis werden diejenigen Entitäten zurückgegeben, die innerhalb bzw. außerhalb eines benutzerdefinierten Abstands liegen.

Semantik: Der `Nearer`-Operator selektiert Entitäten, deren Minimalabstand zueinander unter die durch den `distance`-Parameter angegebene Grenze fällt. Die für den Parameter genutzte Einheit muss in der Ablaufumgebung festgelegt oder aus der referenzierten Instanzdatei abgeleitet werden. Die metrischen Operatoren nutzen das in Abschnitt 6.2.3 vorgestellte Konzept zur Abstraktion von Kontrollstrukturen durch Operatorüberladungen. Daher liegt der `Nearer`-Operator in vier Überladungen vor.

Überladung 1: Die Argumente `set1` und `set2` geben die zu verarbeitenden Datenquellen an. Zwischen allen Entitäten der ersten und zweiten Menge werden deren gegenseitigen Abstände

ermittelt. Dazu werden die räumlichen Repräsentationen der Entitäten genutzt. Falls der Abstand unter den Grenzwert `distance` fällt wird das Entitätspaar in die Ergebnisrelation mit Stelligkeit 2 aufgenommen. Die Signatur der Überladung ist textuell in Auflistung 6.70 und graphisch in Abbildung 6.43 dargestellt.

```
Nearer(set1 : Set, set2 : Set, distance : Float) : Relation
```

Auflistung 6.70: [t]QL4BIM-Signatur des Nearer-Operators, Überladung 1

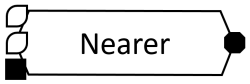


Abbildung 6.43: [v]QL4BIM-Repräsentation des Nearer-Operators, Überladung 1

Auflistung 6.61 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport erfolgen zwei Typfilterungen auf IfcColumn und IfcPipeSegment. Die Mengen an Rohren und Stützen werden in Zeile 4 dem `Nearer`-Operator übergeben. Als Ergebnis liefert die Anfrage die zweistellige Relation `a` zurück. Diese enthält Paare aus Rohren und Stützen, die weniger als 50 Einheiten voneinander entfernt sind.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 columns = TypeFilter(entities is IfcColumn)
3 pipes = TypeFilter(entities is IfcPipeSegment)
4 a[pile|column] = Nearer(piles, columns, 50)
```

Auflistung 6.71: Beispiel zur Nutzung des Nearer-Operators in [t]QL4BIM, Überladung 1

Überladung 2: Die Argumente `relation1` und `relation2` geben die zu verarbeitenden relationalen Datenquellen an. Alle Tupel der ersten Relation werden mit allen Tupeln der zweiten Relation verglichen. Zur Auswahl welche Entitäten aus den Tupeln jeweils auf Unterschreiten des gegenseitigen Abstands hin untersucht werden, dienen die zwei relationalen Argumente `att1` und `att2`. Die Stelligkeit der Ergebnisrelation ist die Summe aus den Stelligkeiten der beiden Eingangsrelationen. Die Attribute von Relation `relation1` werden als erstes in die Ergebnisrelation übernommen. Danach folgen die Attribute von Relation `relation2`. Die Signatur des Operators ist in ihrer textuellen Form in Auflistung 6.72 dargestellt. Abbildung 6.44 zeigt die graphische Repräsentation des Operators.

```
Nearer(relation1 : Relation, att1 : RelAtt,
       relation2 : Relation, att2 : RelAtt, distance : Float) : Relation
```

Auflistung 6.72: [t]QL4BIM-Signatur des Nearer-Operators, Überladung 2

Auflistung 6.73 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem initialen Import liefert der `Deassociator`-Operator durch zweifache Attribut-Dereferenzierung eine zweistel-

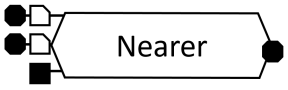


Abbildung 6.44: [v]QL4BIM-Repräsentation des Nearer-Operators, Überladung 2

lige Relation. Der folgende Typfilter selektiert nur die Tupel, deren Entität am Index `1` eine IfcAlarm-Instanz und am Index `2` eine IfcPipeSegment-Instanz darstellen (Z. 3). Diese typbasierte Filterung einer Relation wird für die Typen IfcSensor und IfcPipeSegment wiederholt (Z. 4). In Zeile 5 wird der `Nearer`-Operator mit den Relationen `b` und `c` als Argumente aufgerufen. Für die erste Relation wird das relationale Attribut `alarm` und für die zweite Relation `sensor` ausgewählt. Der Operator liefert eine vierstellige Relation aus Alarmelement, alarmbezogenem Rohrstück, Sensorelement und sensorbezogenem Rohr. Alarm- und Sensorelement sind dabei weniger als 65 Einheiten voneinander entfernt.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[element1|element2] = Deassociator(entities.AssignedToFlowElement)
3 b[alarm|pipe] = TypeFilter(a, [1] is IfcAlarm, [2] is IfcPipeSegment)
4 c[sensor|pipe] = TypeFilter(a, [1] is IfcSensor, [2] is IfcPipeSegment)
5 d[alarm|pipeA|sensor|pipeS]= Nearer(b, b[alarm], c, c[sensor], 65)
```

Auflistung 6.73: Beispiel zur Nutzung des Nearer-Operators in [t]QL4BIM, Überladung 2

Überladung 3: Das Argument `relation` gibt die zu verarbeitenden relationalen Datenquellen an. Zur Spezifikation, welche Entitäten aus dem Tupeln hinsichtlich ihrer Entfernung zueinander untersucht werden sollen, dienen die zwei relationalen Argumente `attribute1` und `attribute2`. Die Stelligkeit der Ergebnisrelation ist mit der Stelligkeit der Eingangsrelation identisch. Die Signatur der Überladung ist textuell in Auflistung 6.74 und graphisch in Abbildung 6.45 dargestellt.

```
Nearer(relation : Relation, att1 : RelAtt, att2 : RelAtt,
       distance : Float) : Relation
```

Auflistung 6.74: [t]QL4BIM-Signatur des Nearer-Operators, Überladung 3

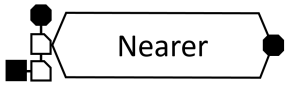


Abbildung 6.45: [v]QL4BIM-Repräsentation des Nearer-Operators, Überladung 3

Auflistung 6.75 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und zweifacher Attribut-Dereferenzierung von `entities.AssignedToFlowElement` wird ein Typfilter aufgerufen. Dieser übernimmt nur die Tupel, deren Entität am Index `1` ein IfcAlarm und am Index `2` ein IfcPipeSegment darstellten (Z. 3). In Zeile 4 wird der `Nearer`-Operator mit der Relation `b` als Argument aufgerufen. Als relationale Attribut wird der erste und der zweite Index aus-

gewählt. Der Operator liefert eine zweistellige Relation aus Alarmelement und assoziiertem Rohrstück zurück. Alarmelement und Rohrstück sind dabei weniger als 65 Einheiten voneinander entfernt.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[element1|element2] = Deassociator(entities.AssignedToFlowElement)
3 b[alarm|pipe] = TypeFilter(a, [1] is IfcAlarm, [2] is IfcPipeSegment)
4 c[alarm|pipe] = Nearer(b, [1], [2], 65)
```

Auflistung 6.75: Beispiel zur Nutzung des Nearer-Operators in [t]QL4BIM, Überladung 3

Überladung 4: Diese Überladung verhält sich wie Überladung 3 und untersucht stets das erste und zweite relationale Attribut der übergebenen Relation. Die Signatur des Operators ist in ihrer textuellen Form in Auflistung 6.76 dargestellt. Abbildung 6.46 zeigt die graphische Repräsentation des Operators.

```
Nearer(relation : Relation, distance : Float) : Relation
```

Auflistung 6.76: [t]QL4BIM-Signatur des Nearer-Operators, Überladung 4

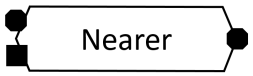


Abbildung 6.46: [v]QL4BIM-Repräsentation des Nearer-Operators, Überladung 4

Auflistung 6.77 zeigt ein Beispiel zur Nutzung der Überladung. Das Selektionsergebnis ist hierbei identisch mit dem vorherigen Beispiel und zeigt die Auswertung des ersten und zweiten Index als Default des Operators.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[element1|element2] = Deassociator(entities.AssignedToFlowElement)
3 b[alarm|pipe] = TypeFilter(a, [1] is IfcAlarm, [2] is IfcPipeSegment)
4 c[alarm|pipe] = Nearer(b, 65)
```

Auflistung 6.77: Beispiel zur Nutzung des Nearer-Operators in [t]QL4BIM, Überladung 4

Die direktionalen Operatoren Above, Below, North, South, West, East

Die direktionalen Operatoren in QL4BIM stützen sich auf die Definitionen von Borrmann (2007, Kap.7.4.3). Die benötigten Algorithmen zur Umsetzung der direktionalen Prädikate wurden in Kapitel 5.2.2 diskutiert. Die Operatoren liegen jeweils in einer strikten und einer relaxierten Variante vor. Somit stellt die Anfragesprache die Operatoren `AboveStrict`, `BelowStrict`, `NorthStrict`, `SouthStrict`, `WestStrict` und `EastStrict` bereit. Zur rela-

xierten Auswertung dienen die Operatoren `AboveRelaxed`, `BelowRelaxed`, `NorthRelaxed`, `SouthRelaxed`, `WestRelaxed` und `EastRelaxed`. Wegen der Ähnlichkeit in der Anwendung werden die direktionalen Operatoren ausschließlich am Beispiel des `AboveStrict`-Operators erläutert.

Semantik: Der `AboveStrict`-Operator selektiert Entitäten, die komplett über einer Referenzentität lokalisiert sind.

Überladung 1: Die Argumente `set1` und `set2` geben die zu verarbeitenden Datenquellen an. Aus den Entitäten beider Mengen werden die Paare selektiert, bei denen sich die zweite Entität komplett über der ersten Entität befindet. Dazu werden die räumlichen Repräsentationen der Entitäten genutzt. Die Signatur der Überladung ist textuell in Auflistung 6.78 und graphisch in Abbildung 6.47 dargestellt.

```
AboveStrict(set1 : Set, set2 : Set) : Relation
```

Auflistung 6.78: [t]QL4BIM-Signatur des AboveStrict-Operators, Überladung 1

Abbildung 6.47: [v]QL4BIM-Repräsentation des AboveStrict-Operators, Überladung 1

Auflistung 6.79 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport werden Stützen und Rohren durch Typfilterung selektiert. In Zeile 4 werden diese Mengen dem `AboveStrict`-Operator übergeben. Als Ergebnis liefert die Anfrage die zweistellige Relation `a` zurück. Diese enthält Paare aus Stützen und Rohren, wobei sich das Rohr komplett über der Stützte befindet.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 columns = TypeFilter(entities is IfcColumn)
3 pipes = TypeFilter(entities is IfcPipeSegment)
4 a[column|pile] = AboveStrict(columns, piles)
```

Auflistung 6.79: Beispiel zur Nutzung des AboveStrict-Operators in [t]QL4BIM, Überladung 1

Überladung 2: Die Argumente `relation1` und `relation2` geben die zu verarbeitenden relationalen Datenquellen an. Alle Tupel der ersten Relation werden mit allen Tupeln der zweiten Relation verglichen. Zur Auswahl, welche Entitäten aus den Tupeln jeweils auf direktionale Überlagerung hin untersucht werden, dienen die zwei relationalen Argumente `att1` und `att2`. Die Stelligkeit der Ergebnisrelation ist die Summe aus den Stelligkeiten der beiden Eingangsrelationen. Die Attribute von Relation `relation1` werden als erstes in die Ergebnisrelation übernommen. Danach folgen die Attribute von Relation `relation2`. Die Signatur des Ope-

rators ist in ihrer textuellen Form in Auflistung 6.80 dargestellt. Abbildung 6.48 zeigt die graphische Repräsentation des Operators.

```
AboveStrict(relation1 : Relation, att1 : RelAtt,
            relation2 : Relation, att2 : RelAtt) : Relation
```

Auflistung 6.80: [t]QL4BIM-Signatur des AboveStrict-Operators, Überladung 2

Abbildung 6.48: [v]QL4BIM-Repräsentation des AboveStrict-Operators, Überladung 2

Auflistung 6.81 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und zweifacher Attribut-Dereferenzierung von `entities.ContainedInStructure` wird ein `TypeFilter`-Operator aufgerufen. Durch diesen werden nur die Tupel übernommen, deren Entität am Index `1` ein IfcSlab und am Index `2` ein IfcBuildingStorey darstellen (Z. 3). Diese typbasierte Filterung einer Relation wird mit IfcCovering und IfcSpace nochmals durchgeführt (Z. 4). In Zeile 5 wird der `AboveStrict`-Operator mit den Argumenten `b` und `c` aufgerufen. Für die Relation `b` wird das relationale Attribut `storey`, und für die Relation `c space` ausgewählt. Der Operator liefert eine vierstellige Relation aus Platte, deren Stockwerk, Belag und dessen Raum zurück. Der Raum befindet sich dabei komplett über dem Stockwerk.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[element1|element2] = Deassociator(entities.ContainedInStructure)
3 b[slab|storey] = TypeFilter(a, [1] is IfcSlab, [2] is IfcBuildingStorey)
4 c[covering|space] = TypeFilter(a, [1] is IfcCovering, [2] is IfcSpace)
5 d[slab|storey|covering|space]= AboveStrict(b, b[storey], c, c[space])
```

Auflistung 6.81: Beispiel zur Nutzung des Nearer-Operators in [t]QL4BIM, Überladung 2

Überladung 3: Das Argument `relation` gibt die zu verarbeitenden relationalen Datenquellen an. Aus allen Tupel der Relation werden zwei Entitäten ausgewählt und untersucht. Zur Spezifikation, welche Entitäten aus dem Tupeln hinsichtlich ihrer direktionalen Überlagerung zueinander untersucht werden sollen, dienen die zwei relationalen Argumente `att1` und `att2`. Die Stelligkeit der Ergebnisrelation ist mit der der Eingangsrelation identisch. Die Signatur der Überladung ist textuell in Auflistung 6.82 und graphisch in Abbildung 6.49 dargestellt.

```
AboveStrict(relation : Relation, att1 : RelAtt, att2 : RelAtt)
            : Relation
```

Auflistung 6.82: [t]QL4BIM-Signatur des AboveStrict-Operators, Überladung 3

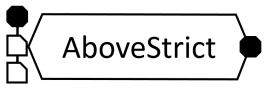


Abbildung 6.49: [v]QL4BIM-Repräsentation des AboveStrict-Operators, Überladung 3

Auflistung 6.83 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Enitätsimport wird zweimal der `Deassociator`-Operator aufgerufen. Dabei wird das *ProvidesBoundaries*- und das *ConnectedTo*-Attribut ausgewertet. Somit erhält man eine dreistellige Relation aus führender Entität, räumlicher Struktur und verbundener Entität. Durch den folgenden Typfilter werden nur die Tupel übernommen, deren Entität am Index `1` ein IfcSlab und am Index `3` eine IfcWall darstellen (Z. 4). In Zeile 5 wird der `AboveStrict`-Operator mit der Relation `c` als Argument aufgerufen. Als relationale Attribut wird der erste und der dritte Index ausgewählt. Der Operator liefert eine dreistellige Relation aus Platte, assoziierter räumlicher Struktur und verbundener Wand zurück. Die Wand ist komplett über der Platte lokalisiert.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|boundary] = Deassociator(entities.ProvidesBoundaries)
3 b[entity|boundary|element] = Deassociator(a, [1].ConnectedTo)
4 c[slab|boundary|wall] = TypeFilter(b, [1] is IfcSlab, [3] is IfcWall)
5 d[slab|boundary|wall] = AboveStrict(c, [1], [3])
```

Auflistung 6.83: Beispiel zur Nutzung des AboveStrict-Operators in [t]QL4BIM, Überladung 3

Überladung 4: Diese Überladung verhält sich wie Überladung 3 und untersucht stets das erste und zweite relationale Attribut der übergebenen Relation. Die Signatur der Überladung ist textuell in Auflistung 6.84 und graphisch in Abbildung 6.50 dargestellt.

```
AboveStrict(relation : Relation) : Relation
```

Auflistung 6.84: [t]QL4BIM-Signatur des AboveStrict-Operators, Überladung 4

Abbildung 6.50: [v]QL4BIM-Repräsentation des AboveStrict-Operators, Überladung 4

Auflistung 6.85 zeigt ein Beispiel zur Nutzung des `AboveStrict`-Operators in Überladung 4. Nach dem initialen Import erfolgen zwei Typfilterungen (Z. 2 u. 3). Die erhaltenen Platten und Stützen werden in Zeile 4 auf Berührung getestet. Das relationale Ergebnis der topologischen Auswertung wird in Zeile 5 dem `AboveStrict`-Operator übergeben. Selektiert werden hierbei nur die Platten-Stützen-Paare, in denen sich die Stütze komplett über der Platte befindet.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities is IfcSlab)
3 columns = TypeFilter(entities is IfcColums)
4 a[slab|column] = Touches(slabs, columns)
5 b[slab|column] = AboveStrict(a)
```

Auflistung 6.85: Beispiel zur Nutzung des AboveStrict-Operators in ^{t}QL4BIM, Überladung 4

6.6 Sekundäre Operatoren

Ein sekundärer Operator nutzt intern andere Operatoren der Anfragesprache. Konzeptionell werden sekundäre Operatoren in QL4BIM eingesetzt, um die Analyse IFC-basierter Daten effektiver zu gestalten. Dies ist besonders für die Verarbeitung hochstrukturierter Bereiche des IFC-Schemas sinnvoll, wie beispielsweise bei IfcPropertySets und objektivierten Relationen. Auch die Analyse zeitlicher Aspekte der Datenbasis erfolgt auf Basis sekundärer Operatoren. Entsprechende intervallbasierten Operatoren wurden in Kapitel 5.1 entwickelten. Der Abschnitt endet mit der Betrachtung der Operatoren zur Teilmodellextraktion und Modellreintegration.

6.6.1 Spezielle Dereferenzierungsoperatoren

Innerhalb dieses Abschnitts werden sekundäre Operatoren vorgestellt, die auf der wiederholten Dereferenzierung von Entitätsattributen basieren. Dazu gehören der `Deassociater`-Operator zur Auflösung von objektivierten Relationen und der `TimeResolver`-Operator zu Dereferenzierung aufgabenbezogener zeitlicher Daten. Der `ProperatyFilter`-Operator kombiniert die Dereferenzierung zwischen Entitäten und deren Eigenschaftsmengen mit einer attributiven Filterung.

Deassociater-Operator

Semantik: Der `Deassociater`-Operator löst objektivierte Beziehungen des IFC-Schemas auf. Die diesbezügliche Modellierung wurde in Kapitel 3.3.4 erläutert. Der Operator navigiert von einer ausgehenden Entität durch die Nennung eines Entitätsattributs zum objektivierten Beziehungsobjekt. Ein derartiges Objekt beinhaltet stets zwei Attribute zur Realisierung der Beziehung. Der Operator nutzt dieses Muster und navigiert ohne explizite Auszeichnung des zweiten Attributs zur verknüpften Entität.

Zur Verdeutlichung zeigt Auflistung 6.86 die Schemadefinition der objektivierten Beziehung IfcRelVoidsElement. Die zwei relevanten Attribute der Entität sind *RelatingBuildingElement*

und *RelatedOpeningElement*. In IfcElement ist *HasOpenings* als inverses Attribut für *RelatingBuildingElement* angegeben. Bei der Nutzung von *HasOpenings* navigiert man somit zum IfcRelVoidsElement. Daraufhin wird stets das **ungenutzte** Attribut, hier also *RelatedOpeningElement*, dereferenziert.

```
1  ENTITY IfcRelVoidsElement
2  ...
3  RelatingBuildingElement : IfcElement;
4  RelatedOpeningElement : IfcFeatureElementSubtraction;
5  END_ENTITY;
6
7  ENTITY IfcElement
8  ...
9  HasOpenings : SET [0:?] OF IfcRelVoidsElement FOR RelatingBuildingElement;
10 END_ENTITY;
```

Auflistung 6.86: Schemadefinition der objektivierten Beziehung IfcRelVoidsElement und das passende inverse Attribut in IfcElement, Überladung 4

Intern genutzte Operatoren: `Dereferencer`

Überladung 1: In den ersten Überladungen wird ein mengenbasierter Attributzugriff als Argument übergeben. Dieser besteht aus einer Set-Variable, dem Punkt-Operator und einem externen Bezeichner für das aufzulösende Entitätsattribut. Als Ergebnis liefert die Überladung eine zweistellige Relation zurück. Die Signatur der Überladung ist textuell in Auflistung 6.87 und graphisch in Abbildung 6.51 dargestellt.

```
Deassociater(setAttributeAccess : SetAttributeAccess) : Relation
```

Auflistung 6.87: [t]QL4BIM-Signatur des Deassociater-Operators, Überladung 1

Abbildung 6.51: [v]QL4BIM-Repräsentationen des Deassociater-Operators, Überladung 1

Auflistung 6.88 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird in Zeile 3 der `Deassociater`-Operator aufgerufen. Als Datenquelle dient die mengenbasierte Variable `elements`. Durch den Attributzugriff wird die *HasOpenings*-Eigenschaft jeder enthaltenen Entität dereferenziert. Danach erfolgt die Dereferenzierung der zweiten Eigenschaft der objektivierten Beziehung. Das Beziehungsobjekt wird nicht in das Ergebnis mit aufgenommen. Die Anfrage liefert somit eine zweistellige Relation aus Elementen und Öffnungen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 elements = TypeFilter(entities is IfcElement)
3 a[element|opening] = Deassociater(elements.HasOpenings)
```

Auflistung 6.88: Beispiel zur Nutzung des Deassociater-Operators in ᵗQL4BIM, Überladungen 1

Überladung 2: In dieser Überladung wird eine Relation und ein relationaler Attributzugriff übergeben. Dieser besteht aus einem relationalen Attribut, dem Punkt-Operator und einem externen Bezeichner für das zu dereferenzierende Entitätsattribut. Als Ergebnis liefert die Überladung eine Relation zurück. Diese ist identisch mit der Eingangsrelation, enthält aber ein zusätzliches Attribut. Die Stelligkeit der Ergebnisrelation ist damit um eins höher als die der Eingangsrelation. Das zusätzliche Attribut beinhaltet die durch die zweite Dereferenzierung gewonnenen Entitäten. Die Signatur der Überladung ist textuell in Auflistung 6.89 und graphisch in Abbildung 6.52 dargestellt.

```
Deassociater(relation: Relation, attributeAccess : RelAttributeAccess)
            : Relation
```

Auflistung 6.89: ᵗQL4BIM-Signatur des Deassociater-Operators, Überladung 2

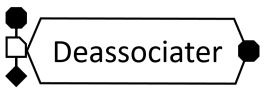


Abbildung 6.52: ᵛQL4BIM-Repräsentationen des Deassociater-Operators, Überladung 2

Auflistung 6.90 zeigt ein Beispiel zur Nutzung der Überladung. Nach Import und Typfilterung wird mittels `Deassocitater`-Operator eine Relation aus Elementen und von diesen gefüllten Öffnungen gewonnen (Z. 3). Diese Relation wird in Zeile 4 dem zweiten `Deassociater`-Operator übergeben. Dieser navigiert von der Öffnung zum objektivierten Beziehungsobjekt IfcRelVoidsElement und löst dort das Attribut *RelatingBuildingElement* auf. Somit erhält man eine dreistellige Relation aus füllendem Bauteil, Öffnung und befülltem Bauteil.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 elements = TypeFilter(entities is IfcElement)
3 a[element|opening] = Deassociater(elements.FillsVoids)
4 b[element|opening|host] = Deassociater(a, [opening].VoidsElements)
```

Auflistung 6.90: Beispiel zur Nutzung des Deassociater-Operators in ᵗQL4BIM, Überladungen 2

TimeResolver-Operator

Semantik: In der IFC-basierten Modellierung wird eine Aufgabe als IfcTask repräsentiert und kann mit Bauteilen (IfcElements) verknüpft werden. Darauf wurde im Kapitel 3.5.3 genauer eingegangen. Die zeitlichen Aspekte der Aufgabe werden in der verknüpften IfcTaskTime-Entität vorgehalten. Der `TimeResolver`-Operator löst die objektivierten Beziehungen zwischen einem Bauteil und der Aufgabe auf und liefert Tupel aus Bauteil sowie IIfcTaskTime zurück. Ist kein IfcTask und IfcTaskTime mit dem Bauteil verknüpft, wird kein Tupel erstellt.
Intern genutzte Operatoren: `Dereferencer` und `Deassociater`
Überladung 1: In dieser Überladung wird ausschließlich ein mengenbasiertes Argument übergeben. Für jede enthaltene Entität wird eine Dereferenzierung aller verknüpfter IfcTask-Entitäten durchgeführt. Danach erfolgt eine Dereferenzierung aller IfcTaskTime-Entitäten. Zurückgegeben wird für jede Eingangsentität der jeweilige IfcTask und die IfcTaskTime. Somit liefert die Überladung eine Relation mit der Stelligkeit 3. Die Signatur der Überladung ist textuell in Auflistung 6.91 und graphisch in Abbildung 6.53 dargestellt.

```
TimeResolver(set : Set) : Relation
```

Auflistung 6.91: [t]QL4BIM-Signatur des TimeResolver-Operators, Überladung 1

Abbildung 6.53: [v]QL4BIM-Repräsentation des TimeResolver-Operators, Überladung 1

Auflistung 6.92 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Entitätsimport und einer Typfilterung mit IfcWindow wird in Zeile 3 der `TimeResolver`-Operator aufgerufen. Dieser dereferenziert für jede übergebene Fenster-Entität deren verknüpfte IfcTasks und IfcTaskTimes. Die gewonnene dreistellige Relation wird der Variable `a[window|task|time]` zugewiesen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 windows = TypeFilter(entities is IfcWindow)
3 a[window|task|time] = TimeResolver(windows)
```

Auflistung 6.92: Beispiel zur Nutzung des TimeResolver-Operators in [t]QL4BIM, Überladungen 1

Überladung 2: Diese Überladung verhält sich identisch zur ersten Überladung. Durch ein zusätzliches Argument werden nur IfcTasks mit einem bestimmten *Name*-Attribute selektiert. Die Signatur der Überladung ist textuell in Auflistung 6.93 und graphisch in Abbildung 6.54 dargestellt.

```
TimeResolver(set : Set, name : String) : Relation
```

Auflistung 6.93: [t]QL4BIM-Signatur des TimeResolver-Operators, Überladung 2

Abbildung 6.54: [v]QL4BIM-Repräsentation des TimeResolver-Operators, Überladung 2

Auflistung 6.94 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Import von Entitäten und einer Typfilterung mit IfcWall wird in Zeile 3 der `TimeResolver`-Operator aufgerufen. Dieser dereferenziert für jede übergebene Wand-Entität deren verknüpfte IfcTasks. Selektiert und weiter dereferenziert werden nur IfcTask-Instanzen deren *Name*-Attribut dem Argument `name` entspricht. Die gewonnene dreistellige Relation wird der Variable `a[wall|task|time]` zugewiesen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 walls = TypeFilter(entities is IfcWall)
3 a[wall|task|time] = TimeResolver(walls, "concreting")
```

Auflistung 6.94: Beispiel zur Nutzung des TimeResolver-Operators in [t]QL4BIM, Überladungen 2

Überladung 3: In dieser Überladung wird eine Relation als Datenquelle übergeben. Das relationale Attribut wird durch `attribute` angegeben. Für den entsprechenden Index jedes übergebenen Tupels wird eine Dereferenzierung aller verknüpfter IfcTasks durchgeführt. Danach erfolgt eine Dereferenzierung aller IfcTaskTime-Entitäten. Zurückgegeben werden für jedes Eingangstupel der jeweilige IfcTask und die IfcTaskTime. Somit liefert die Überladung eine Relation zurück, deren Stelligkeit um 2 höher ist als die der Eingangsrelation. Die Signatur der Überladung ist textuell in Auflistung 6.95 und graphisch in Abbildung 6.55 dargestellt.

```
TimeResolver(relation : Relation, attribute : RelAtt) : Relation
```

Auflistung 6.95: [t]QL4BIM-Signatur des TimeResolver-Operators, Überladung 3

Abbildung 6.55: [v]QL4BIM-Repräsentation des TimeResolver-Operators, Überladung 3

Auflistung 6.96 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Import von Entitäten und zwei Typfilterungen (IfcSlab u. IfcWall) wird in Zeile 3 der `Touches`-Operator aufgerufen. Die dadurch erhaltenen Paare aus sich berührenden Platten und Wänden werden dem `TimeResolver`-Operator übergeben (Z. 4). Dadurch erhält man eine Relation mit der Stelligkeit 4. Diese besteht aus sich berührenden Platten und Wänden, wandbezogenen Aufgaben und Zeiten.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities is IfcSlab)
3 walls = TypeFilter(entities is IfcWall)
4 a[slab|wall] = Touches(slabs, walls)
5 b[slab|wall|wtask|wtime] = TimeResolver(a, a[wall])
```

Auflistung 6.96: Beispiel zur Nutzung des TimeResolver-Operators in [t]QL4BIM, Überladungen 3

Überladung 4: Diese Überladung verhält sich identisch zur dritten Überladung. Durch ein zusätzliches Argument werden nur IfcTasks mit einem bestimmten *Name*-Attribut selektiert. Die Signatur des Operators ist in ihrer textuellen Form in Auflistung 6.97 und in ihrer graphischen Repräsentation in Abbildung 6.56 dargestellt. Aufgrund der Ähnlichkeit zur Überladung 2 und 3 wird hier auf ein weiteres Beispiel zur Nutzung der Überladung verzichtet.

```
TimeResolver(relation : Relation, attribute : RelAtt, name : String)
            : Relation
```

Auflistung 6.97: [t]QL4BIM-Signatur des TimeResolver-Operators, Überladung 4

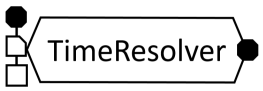


Abbildung 6.56: [v]QL4BIM-Repräsentation des TimeResolver-Operators, Überladung 4

PropertyFilter-Operator

Semantik: Der `PropertyFilter`-Operator selektiert IFC-Entitäten, die einem Theta-Prädikat genügen. Im Gegensatz zum `AttributeFilter`-Operator werden hierbei nicht die direkten Entitätseigenschaften betrachtet. Stattdessen wertet der Operator die mit der Entität verknüpften IfcPropertySets aus. Derartige Eigenschaftsmengen können sowohl einem übergeordneten Typobjekt (IfcTypeObject) oder der eigentlichen Entitätsinstanz zugeordnet sein. Der `PropertyFilter` beachtet diese typbasierten Eigenschaften unter Wahrung der lokalen Eigenschaftspriorität (siehe Tabelle 3.2 in Kapitel 3).
Intern genutzte Operatoren: `Deassociator` und `AttributeFilter`
Überladung 1: In dieser Überladung wird die Datenquelle über die set-Variable innerhalb eines mengenbasierten Attributprädikats festgelegt. Das Prädikat enthält außerdem die zu untersuchende Property. Als Ergebnis wird eine Menge zurückgegeben. Diese beinhaltet nur Entitäten, die über eine Merkmalsliste mit dem genannten Property versehen sind. Außerdem muss der Wert des Merkmals dem Prädikat genügen. Die Signatur der Überladung ist textuell in Auflistung 6.98 und graphisch in Abbildung 6.57 dargestellt.

```
PropertyFilter(predicate : SetAttributePredicate) : Set
```

Auflistung 6.98: tQL4BIM-Signatur des PropertyFilter-Operators, Überladung 1

Abbildung 6.57: vQL4BIM-Repräsentation des PropertyFilter-Operators, Überladung 1

Auflistung 6.99 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Import von Entitäten und einer Typfilterung wird in Zeile 3 der `PropertyFilter`-Operator aufgerufen. Dieser dereferenziert für jede übergebene Treppen-Entität alle Merkmalslisten und sucht in diesen das Property *HasDrive*. Ist diese wahr, wird die Entität in die Ergebnismenge aufgenommen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 stairs = TypeFilter(entities is IfcStair)
3 movingStairs  = PropertyFilter(stairs.HasDrive = true)
```

Auflistung 6.99: Beispiel zur Nutzung des PropertyFilter-Operators in tQL4BIM, Überladungen 1

Überladung 2: Diese Überladung verhält sich identisch zur ersten Überladung. Durch ein zusätzliches Argument werden nur IfcPropertySets mit einem bestimmten *Name*-Attribut selektiert. Die Signatur der Überladung ist textuell in Auflistung 6.100 und graphisch in Abbildung 6.58 dargestellt.

```
PropertyFilter(set: Set, predicate : AttPredicate, name : String) : Set
```

Auflistung 6.100: tQL4BIM-Signatur des PropertyFilter-Operators, Überladung 2

Abbildung 6.58: vQL4BIM-Repräsentation des PropertyFilter-Operators, Überladung 2

Auflistung 6.101 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Import von Entitäten und einer Typfilterung wird in Zeile 3 der `PropertyFilter`-Operator aufgerufen. Dieser dereferenziert für jede übergebene Treppen-Entität alle Merkmalslisten deren *Name*-Attribut `Pset_Condition` entspricht. In diesen wird das Property *AssessmentCondition* gegenüber einer numerischen Schranke geprüft. Wird diese unterschritten wird die Entität in die Ergebnismenge `poorCoverings` aufgenommen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 coverings = TypeFilter(entities is IfcCovering)
3 poorCoverings  = PropertyFilter(coverings.AssessmentCondition < 7,
                                  "Pset_Condition")
```

Auflistung 6.101: Beispiel zur Nutzung des PropertyFilter-Operators in tQL4BIM, Überladungen 2

Überladung 3: In dieser Überladung wird eine Relation als Datenquelle übergeben. Das zu untersuchende Property wird über ein Attributprädikat festgelegt. Für den entsprechenden Index jedes übergebenen Tupels wird eine Dereferenzierung aller verknüpfter IfcPropertySet-Entitäten durchgeführt. Ist in einer Merkmalsliste das genannte Property vorhanden und genügt dieses dem Prädikat, wird das Tupel in die Ergebnisrelation übernommen. Die Stelligkeit von Eingangs- und Ergebnisrelation sind somit identisch. Die Signatur der Überladung ist textuell in Auflistung 6.102 und graphisch in Abbildung 6.59 dargestellt..

```
PropertyFilter(relation : Relation, predicate : AttPredicate) : Relation
```

Auflistung 6.102: tQL4BIM-Signatur des PropertyFilter-Operators, Überladung 3

Abbildung 6.59: vQL4BIM-Repräsentation des PropertyFilter-Operators, Überladung 3

Auflistung 6.103 zeigt ein Beispiel zur Nutzung des Operators. Nach dem Import von Entitäten und einer Typfilterung wird in Zeile 3 der `TimeResolver`-Operator aufgerufen. Die dadurch erhaltene Relation aus Pfählen, pfahlbezogenen Aufgaben und Zeiten wird dem `PropertyFilter`-Operator übergeben (Z. 4). In diesem wird das Property *Status* des ersten relationalen Attributs untersucht. Dadurch erhält man eine Relation aus Pfählen, pfahlbezogenen Aufgaben und Zeiten, in denen jeder Pfahl den *Status* `NEW` aufweist.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 piles = TypeFilter(entities is IfcPile)
3 a[pile|task|time] = TimeResolver(piles)
4 b[newPile|task|time]  = PropertyFilter(a, [pile].Status = "NEW")
```

Auflistung 6.103: Beispiel zur Nutzung des PropertyFilter-Operators in tQL4BIM, Überladungen 3

Überladung 4: Diese Überladung verhält sich identisch zur dritten Überladung. Durch ein zusätzliches Argument werden nur IfcPropertySets mit einem bestimmten *Name*-Attribut selektiert. Die Signatur der Überladung ist textuell in Auflistung 6.104 und graphisch in Abbildung 6.60 dargestellt. Aufgrund der Ähnlichkeit zur Überladung 2 und 3 wird hier auf ein weiteres Beispiel zur Nutzung der Überladung verzichtet.

```
PropertyFilter(relation : Relation, predicate : AttPredicate, name : String)
              : Relation
```

Auflistung 6.104: [t]QL4BIM-Signatur des PropertyFilter-Operators, Überladung 4

Abbildung 6.60: [v]QL4BIM-Repräsentation des PropertyFilter-Operators, Überladung 4

6.6.2 Zeitliche Operatoren

Neben den räumlich-topologischen Operatoren werden in QL4BIM auch zeitlich-topologische Operatoren angeboten. Diese sind für die temporale Analyse der Bauablaufplanung konzipiert und basieren auf dem Konzept von Vilain (1982). Auf die hierfür relevante BIM-basierte Bauablaufplanung wurde in Kapitel 3.5 eingegangen. Das zeitliche Modell von Vilain und die daraus abgeleiteten Operatoren für die Bauablaufplanung wurden in Kapitel 5.1 erörtert.

Die Anfragesprache umfasst die neun domänenspezifischen, zeitlich-topologischen Operatoren `Before`, `After`, `Meets`, `Encloses`, `During`, `Starts`, `Ends`, `Overlays` und `Identical`. Die Operatoren nutzen teilweise das in Abschnitt 6.2.3 beschriebene Konzept zur Abstraktion von Kontrollstrukturen durch Operatorüberladungen. Da die Operatorengruppe ausschließlich Relationen verarbeiten, wird die erste Überladungsvariante mit zwei mengenbasierten Variablen nicht angeboten. Auch Standardüberladung 4 wird nicht umgesetzt, da die Indizes der zeitlichen Daten tendenziell variieren. Wegen der Ähnlichkeit in der Anwendung wird die Operatorengruppe ausschließlich am Beispiel des `Meets`-Operators erläutert.

Semantik: Der `Meets`-Operator selektiert Entitäten, deren aufgabenbezogene, zeitliche Daten zwei sich berührende Intervalle darstellen. Die Intervalldaten müssen als IfcTaskTime übergeben werden.
Intern genutzte Operatoren: `AttributeFilter`
Überladung 1: Die Argumente `relation1` und `relation2` geben die zu verarbeitenden relationalen Datenquellen an. Alle Tupel der ersten Relation werden mit allen Tupeln der zweiten Relation verglichen. Zur Auswahl, welche Entitäten aus den Tupeln jeweils auf zeitliche Berührung hin untersucht werden, dienen die zwei relationalen Argumente `attribute1` und `attribute2`. Die Stelligkeit der Ergebnisrelation ist die Summe aus den Stelligkeiten der beiden Eingangsrelationen. Die Attribute von Relation `relation1` werden als erstes in die Ergebnisrelation übernommen. Danach folgen die Attribute von Relation `relation2`. Die Signatur der Überladung ist textuell in Auflistung 6.105 und graphisch in Abbildung 6.64 dargestellt.

```
Meets(relation1 : Relation, attribute1 : RelAtt,
      relation2 : Relation, attribute2 : RelAtt) : Relation
```

Auflistung 6.105: [t]QL4BIM-Signatur des Meets-Operators, Überladung 1

Abbildung 6.61: [v]QL4BIM-Repräsentation des Meets-Operators, Überladung 1

Auflistung 6.106 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Import erfolgen zwei Typfilterungen mit IfcSlab und IfcWall. Durch die zwei folgenden `TimeResolver`-Operatoren werden die jeweiligen aufgabenbezogenen zeitlichen Daten (IfcTask u. IfcTaskTime) sowohl für Platten als auch für Wände dereferenziert (Z. 4 u. 5). In Zeile 6 werden dem `Meets`-Operator die Relationen `a` und `b` übergeben. Auf zeitliche Berührung werden jeweils die Entitäten des dritten Index `time` überprüft. Nach einer Projektion besteht jedes Tupel der Ergebnisrelation `d` aus Platte, plattenbezogenen Zeitdaten, Wand und wandbezogenen Zeitdaten. Die zeitlichen Intervalle berühren sich außerdem.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities is IfcSlab)
3 walls = TypeFilter(entities is IfcWall)
4 a[slab|task|time] = TimeResolver(slabs)
5 b[wall|task|time] = TimeResolver(walls)
6 c[slab|taskS|timeS|wall|taskW|timeW] = Meets(a, a[time], b, b[time])
7 d[slab|timeS|wall|timeW]= Projector(c, [1], [3], [4], [6])
```

Auflistung 6.106: Beispiel zur Nutzung des Meets-Operators in [t]QL4BIM, Überladung 2

Überladung 2: Das Argument `relation` gibt die zu verarbeitenden relationalen Datenquellen an. Aus allen Tupel der Relation werden zwei Entitäten ausgewählt und untersucht. Zur Spezifikation, welche Entitäten aus dem Tupeln auf gegenseitige Berührung hin untersucht werden sollen, dienen die zwei relationalen Argumente `attribute1` und `attribute2`. Die Stelligkeit der Ergebnisrelation ist mit der Stelligkeit der Eingangsrelation identisch. Die Signatur der Überladung ist textuell in Auflistung 6.107 und graphisch in Abbildung 6.62 dargestellt.

```
Meets(relation : Relation, attribute1 : RelAtt, attribute2 : RelAtt)
     : Relation
```

Auflistung 6.107: [t]QL4BIM-Signatur des Meets-Operators, Überladung 2

Auflistung 6.108 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Import und zwei Typfilterungen erfolgt die räumlich-topologische Prüfung mit dem `Touching`-Operator (Z. 4).

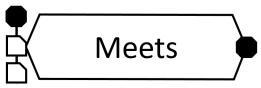


Abbildung 6.62: [v]QL4BIM-Repräsentation des Meets-Operators, Überladung 2

Somit erhält man eine zweistellige Relation aus sich berührenden Platten und Wänden. Durch die zwei folgenden `TimeResolver`-Operatoren werden die jeweiligen aufgabenbezogenen zeitlichen Daten (IfcTask u. IfcTaskTime) dereferenziert (Z. 5 u. 6). In Zeile 7 werden über den `Projector`-Operator die nicht benötigten IfcTasks entfernt. Somit wird dem `Meets`-Operator die vierstellige Relation `d` übergeben. Auf zeitliche Berührung werden die Entitäten des dritten und vierten Index überprüft. Jedes Tupel der Ergebnisrelation `e` besteht aus Platte, plattenbezogenen Zeitdaten, berührender Wand und wandbezogenen Zeitdaten. Die durch die zeitlichen Daten vorgehaltenen Intervalle berühren sich außerdem.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slabs = TypeFilter(entities is IfcSlab)
3 walls = TypeFilter(entities is IfcWall)
4 a[slab|wall] = Touching(slabs, walls)
5 b[slab|wall|taskS|timeS] = TimeResolver(a, [1])
6 c[slab|wall|taskS|timeS|taskW|timeW] = TimeResolver(b, [2])
7 d[slab|wall|timeS|timeW] = Projector(c, [1], [2], [4], [6])
8 e[slab|wall|timeS|timeW] = Meets(d, [3], [4])
```

Auflistung 6.108: Beispiel zur Nutzung des Meets-Operators in [t]QL4BIM, Überladung 2

6.6.3 Operatoren für die Teilmodellextraktion und Modellreintegration

Innerhalb dieses Abschnitts werden sekundäre Operatoren vorgestellt, die zur Teilmodellextraktion und Modellreintegration genutzt werden können. Die entsprechenden Konzepte für diese Operatoren wurde in Kapitel 5 in den Abschnitten 5.3.1 und 5.3.2 entwickelt.

MvdFilter-Operator

Semantik: Der `MvdFilter`-Operator selektiert alle Entitäten die in einem MVD-Dokument angegeben sind und gibt sie als mengenbasiertes Ergebnis zurück. Zusätzlich wird eine Report-Datei geschrieben, die angibt ob alle nötigen Informationen in der übergebenen Datenbasis verfügbar waren.
Intern genutzte Operatoren: `TypeFilter` und `AttributeFilter`
Überladung 1: Über den `entities`-Parameter wird die Menge der zu filternden Entitäten angegeben. Das String-typisierte `path`-Argument gibt Speicherort und Namen des zu nutzen-

den MVD-Dokuments an. Die Signatur des Operators ist textuell in Auflistung 6.109 und graphisch in Abbildung 6.63 dargestellt.

```
MvdFilter(entities : Set, path : String) : Set
```

Auflistung 6.109: [t]QL4BIM-Signatur des MvdFilter-Operators, Überladung 1

Abbildung 6.63: [v]QL4BIM-Repräsentation des MvdFilter-Operators, Überladung 1

Merger-Operator

Semantik: Der `Merger`-Operator führt einen three-way merge aus und gibt eine zusammengeführte Entitätsmenge zurück. Zusätzlich wird eine Report-Datei geschrieben, ob der merge ohne Konflikte ausgeführt werden konnte. Ansonsten werden diese für die entsprechenden Entitäten aufgeführt.
Intern genutzte Operatoren: `TypeFilter`, `AttributeFilter` und `Equals`
Überladung 1: Über den `entitiesD0`-Parameter wird die Menge der ursprünglichen Entitäten, also des base common ancestor angegeben. Die Parameter `entitiesD1` und `entitiesD2` stellen die Entitäten der beiden simultan weiterentwickelten Modellinstanzen dar. Die Signatur des Operators ist textuell in Auflistung 6.110 und graphisch in Abbildung 6.64 dargestellt.

```
Merger(entitiesD0 : Set, entitiesD1 : Set, entitiesD2 : Set) : Set
```

Auflistung 6.110: [t]QL4BIM-Signatur des MvdFilter-Operators, Überladung 1

Abbildung 6.64: [v]QL4BIM-Repräsentation des Merger-Operators, Überladung 1

Beispielhafte Anfragen mit dem MvdFilter- und dem Merger-Operator sind in Kapitel 8.1.3 aufgeführt.

6.7 Beispiele benutzerdefinierter Operatoren

Neben der Nutzung elementarer und sekundärer Operatoren ist es dem Endanwender in [t]QL4BIM möglich, neue Operatoren zu erstellen. Dadurch kann die Sprache je nach Nutzungsausrichtung angepasst werden. Zur Demonstration werden im Folgenden die zwei benutzerdefinierten Operatoren `WallsInBasement` und `NextTaskWithLag` vorgestellt.

WallsInBasement-Operator

Semantik: Der Operator selektiert Wände, die sich im Kellergeschoss (`basement`) des Gebäudes befinden.
Überladung 1: Über die Signatur des Operators wird der formale Parameter `entities` als mengenbasierte Variable definiert. Nun folgen vier Statements nach den bereits bekannten Prinzipien des globalen Anfragenbereichs. Durch die zweifache Dereferenzierung in Zeile 3 erhält man Paare aus Produkt und räumlicher Struktur. Die Typfilterung selektiert Paare der Typen IfcWall und IfcBuildingStorey (Z. 4). Danach wird zusätzlich hinsichtlich des *Name*-Attributs des Stockwerks gefiltert. Da ausschließlich die Menge der im Kellerstockwerk enthaltenen Wände zurückgegeben werden soll, wird die Relation in Zeile 6 entsprechend projiziert. Die Definition des Operators ist in Auflistung 6.111 dargestellt. Abbildung 6.65 zeigt die graphische Repräsentation des Operators.

```
1 func WallsInBasement:WIB(entities)
2 (
3   a[product|structure] = Deassociater(entities.ContainedInSpatialStructure)
4   b[wall|storey] = TypeFilter(a, [product] is IfcWall,
                                   [structure] is IfcBuildingStorey)
5   c[wall|storey] = AttributeFilter(b, [storey].Name = "Basement")
6   walls = Projector(c, [wall])
7 )
```

Auflistung 6.111: Definition des benutzerdefinierten WallsInBasement-Operators

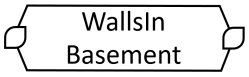


Abbildung 6.65: [v]QL4BIM-Repräsentation des WallsInBasement-Operators

Auflistung 6.112 zeigt ein Beispiel zur Nutzung des benutzerdefinierten Operators. In der ersten Zeile werden alle Entitäten einer Instanzdatei importiert. Danach wird der `WallsInBasement`-Operator aufgerufen und in die mengenbasierte Variable `entities` übergeben. In Zeile 3 wird der `TimeResolver`-Operator aufgerufen. Somit erhält man eine Relation aus Wänden des Kellergeschossen mit ihren aufgabenbezogenen Zeitdaten.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 wibs = WallsInBasement(entities)
3 a[wib|time] = TimeResolver(wibs)
```

Auflistung 6.112: Beispiel zur Nutzung des WallsInBasement-Operators in [t]QL4BIM

NextTaskWithLag-Operator

Semantik: Der Operator verarbeitet eine Relation aus Produkten und Aufgaben. Zurückgegeben werden die Produkte, deren nachfolgende Aufgabe einen zeitlichen Abstand von mindestens einem Tag zur vorhergehenden, übergebenen Aufgabe aufweist.
Überladung 1: Über die Signatur des Operator wird der formale Parameter `a[product|task]` als zweistellige, relationale Variable definiert. Nun folgt eine zweifache Typprüfung um sicher zu stellen, dass ausschließlich IfcProduct-IfcTask-Paare verarbeitet werden. In Zeile 4 wird über `[2].IsPredecessorTo` im `Dereferencer` die objektivierte Relation IfcRelSequence aufgelöst. Durch einen weiteren `Dereferencer` wird die zur Eingangsaufgabe nachfolgende Aufgabe selektiert. In Zeile 6 wird aus der IfcRelSequence-Instanz das *LagValue*-Attribut dereferenziert. Im nachfolgenden Attributfilter wird diese Entität geprüft. Abschließend wird eine Projektion durchgeführt, um Paare aus Produkt und Nachfolgeaufgabe zurückzugeben. Die Nachfolgeaufgabe besitzt hierbei einen zeitlichen Abstand zur Eingangsaufgabe von mindestens einem Tag. Die Definition des Operators ist in Auflistung 6.113 dargestellt. Abbildung 6.66 zeigt die graphische Repräsentation des Operators.

```
1 func NextTaskWithLag:NTL(a[product|task])
2 (
3   b[product|task] = TypeFilter(a, [1] is IfcProduct, [2] is IfcTask)
4   c[product|pre|seq]= Dereferencer(b, [2].IsPredecessorTo)
5   d[product|pre|seq|suc]= Dereferencer(c, [2].RelatedProcess)
6   e[product|pre|seq|suc|lag] = Dereferencer(d, [2].LagValue)
7   f[product|pre|seq|suc|lag] = AttributeFilter(e, [4].TimeLag > "P1D")
8   g[product|successor] = Projector(f, [product], [suc])
9 )
```

Auflistung 6.113: Definition des benutzerdefinierten NextTaskWithLag-Operators

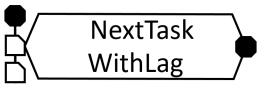


Abbildung 6.66: vQL4BIM-Repräsentation des NextTaskWithLag-Operators

Auflistung 6.114 zeigt ein Beispiel zur Nutzung der Überladung. Nach dem Entitätsimport erfolgt eine Typfilterung auf Fensterinstanzen (IfcWindow). In Zeile 3 werden Aufgaben zu den Fenstern dereferenziert. Diese Fenster-Aufgaben-Relation wird dem `NextTaskWithLag`-Operator übergeben (Z. 4). Somit erhält man eine Relation aus Fenstern und Aufgaben. Die Aufgabe ist der direkte Nachfolger zu Eingangsaufgabe und besitzt einen Mindestabstand von einem Tag zu diesen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 windows = TypeFilter(entities is IfcWindow)
3 a[window|task] = Deassociator(windows.ReferencedBy)
4 b[window|successorOneDayLag] = NextTaskWithLag(a)
```

Auflistung 6.114: Beispiel zur Nutzung des NextTaskWithLag-Operators in tQL4BIM

6.8 Musteranfragen in tQL4BIM und vQL4BIM

Auf Basis der entwickelten elementaren und sekundären Operatoren werden im Folgenden die in Kapitel 4.2 definierten Musteranfragen in QL4BIM umgesetzt. Für eine Erläuterung der einzelnen Verarbeitungsschritte sei auf die Definitionen der genutzten Operatoren in den jeweiligen Überladungen verwiesen.

6.8.1 Musteranfrage 1

Die Anfrage als Langtext lautet: „Selektiere alle Wände.“. Auflistung 6.115 zeigt deren textuelle Umsetzung. Abbildung 6.67 stellt die visuelle Umsetzung mit eingeblendeten, typisierten Konnektoren dar. In Abbildung 6.68 ist die Konnektorendarstellung deaktiviert.

```
entities = ImportModel("C:\TranslaTUM.ifc")
walls = TypeFilter(entities is IfcWall)
```

Auflistung 6.115: Musteranfrage 1 (Wände) in tQL4BIM

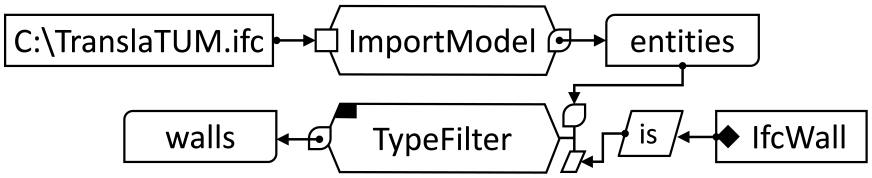


Abbildung 6.67: Musteranfrage 1 (Wände) in vQL4BIM

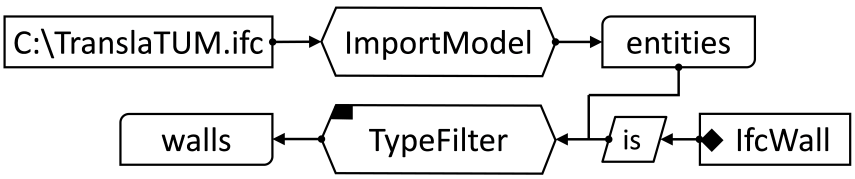


Abbildung 6.68: Musteranfrage 1 (Wände) in vQL4BIM, Konnektorendarstellung deaktiviert

6.8.2 Musteranfrage 2

Die Anfrage als Langtext lautet: „Selektiere Paare aus Wand und beherbergter Tür. Dabei sollen nur Paare berücksichtigt werden, in denen die Tür mindesten 2000 mm hoch ist.“. Auflistung 6.116 und Abbildung 6.69 zeigen deren textuelle und visuelle Umsetzung. Im Anhang in Abbildung A.3 ist die visuelle Anfrage ohne typisierte Konnektoren dargestellt.

```
entities = ImportModel("C:\TranslaTUM.ifc")
a[entity|open] = Deassociater(entities.HasOpenings)
b[entity|open|fill] = Deassociater(a, [open].HasFilling)
c[entity|fill] = Projector(b, [entity], [fill])
d[wall|door] = TypeFilter(c, [entity] is IfcWall, [fill] is IfcDoor)
e[wall|door] = AttributeFilter(d, [door].OverallHeight >= 2000)
```

Auflistung 6.116: Musteranfrage 2 (Wand/Tür Paare) in ᵗQL4BIM

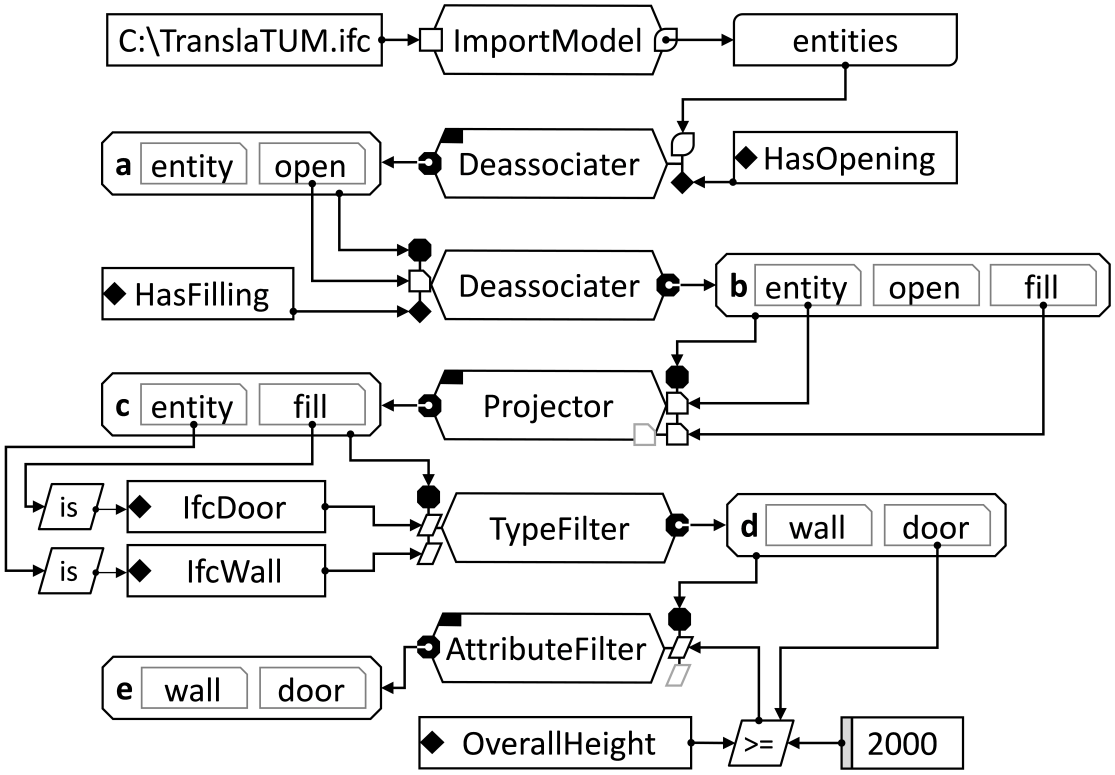


Abbildung 6.69: Musteranfrage 2 (Wand/Tür Paare) in ᵛQL4BIM

6.8.3 Musteranfrage 3

Die Anfrage als Langtext lautet: „Selektiere die Wände, die im Stockwerk mit Namen *First-Floor* enthalten sind und als letztes fertiggestellt werden.“ Auflistung 6.117 und Abbil-

dung 6.70 zeigen deren textuelle und visuelle Umsetzung. Im Anhang in Abbildung A.4 ist die visuelle Anfrage ohne typisierte Konnektoren dargestellt.

```
entities = ImportModel("C:\TranslaTUM.ifc")
a[entity|struc] = Deassociater(entities.ContainedInSpatialStructure)
b[wall|storey] = TypeFilter(a, [1] is IfcWall, [2] is IfcBuildingStorey)
c[wall|storey] = AttributeFilter(b, [storey].Name = "FirstFloor")
d[wall|storey|task|time] = TimeResolver(c, [wall])
e[wall|storey|task|time] = Maximum(d, [time].ActualFinish)
latestWalls = Projector(e, [wall])
```

Auflistung 6.117: Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) in tQL4BIM

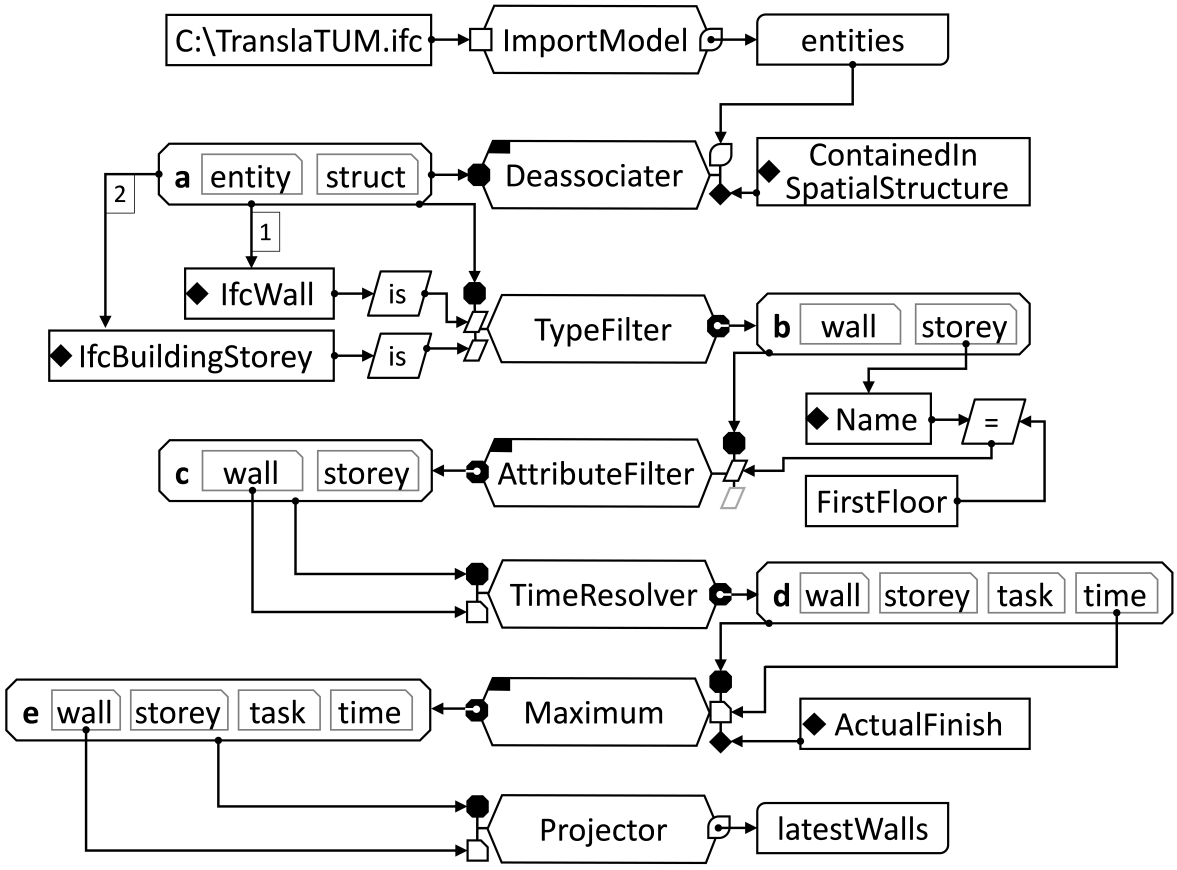


Abbildung 6.70: Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) in vQL4BIM

6.8.4 Musteranfrage 4

Die Anfrage als Langtext lautet: „Selektiere Paare aus Deckenplatte und Wand. Dabei sollen nur Paare berücksichtigt werden, in denen sich Deckenplatte und Wand berühren und sich die Wand komplett unterhalb der Deckenplatte befindet.“.

Die Anfrage wurde bereits zur Erläuterung des sprachlichen Grundprinzips implementiert (siehe Auflistung 6.1). An dieser Stelle sollen die Möglichkeiten zur kompakten Anfragedefinition in QL4BIM gezeigt werden. Auflistung 6.118 beinhaltet die entsprechende Variante der Anfrage. Abbildung 6.71 zeigt die visuelle Umsetzung dieser kompakten Variante von Musteranfrage 4. Im Anhang ist in Abbildung A.5 die kompakte, visuelle Anfrage ohne typisierte Konnektoren dargestellt.

```
e = IM("C:\TranslaTUM.ifc")
s = TF(e is IfcSlab)
w = TF(e is IfcWall)
a[] = TO(s, w)
b[] = BS(a)
```

Auflistung 6.118: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ᵗQL4BIM, kompakte Variante

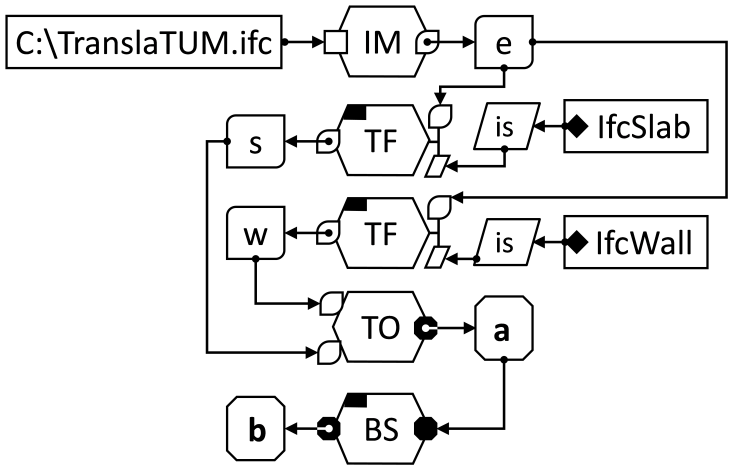


Abbildung 6.71: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ᵛQL4BIM, kompakte Variante

6.9 Zusammenfassung

Dieses Kapitel schildert die Konzeption der Query Language for 4D Building Information Models (QL4BIM). Nach der Betrachtung von etablierten Programmierparadigmen wurden die Anforderungen an eine derartige Anfragesprache festgelegt. Zentrale Charakteristika von QL4BIM sind die Ausrichtung an der imperativen Programmierung und die Unterstützung einer textuellen und einer visuellen Notation. Zur formalen Definition beider Notationen wurden kontextfreie Grammatiken eingeführt.

Die zentrale Einheit der Untersuchung mit QL4BIM ist die Entität. Diese stammt aus einem externen Schema wie der IFC oder CityGML. Die Anfragesprache übernimmt diese Definitionen. Entitäten werden zum einen als komplexe Datenobjekte mit Attributen betrachtet und gleichzeitig als räumliche Repräsentationen verstanden. Diese Vielgestaltigkeit von Entitäten ermöglicht die Bereitstellung einheitlicher Operatoren für räumliche und nicht-räumliche Analysen. Das Vorgehen wurde nicht für zeitliche Operatoren wiederholt. Im Gegensatz zur räumlichen Repräsentation, bei der stets die detaillierteste Variante gewählt werden kann, referenziert eine semantische Entität mitunter unterschiedliche zeitliche Daten, die separat betrachtet und ausgewertet werden müssen.

Die Anfragesprache stellt ein eigenes Typsystem für Aggregationen bereit. Somit können Mengen und Relationen aus Entitäten gebildet werden. Die Operatoren der Anfragesprache verarbeiten derartige Entitätsmengen und -relationen auf hohem Abstraktionsniveau. Auf den expliziten Einsatz von Kontrollstrukturen wie Wenn-Dann-Abfragen und Schleifen wurde bewusst verzichtet. Diese sind in unterschiedlichen Formen in Operatorüberladungen integriert.

Operatoren liegen als elementare, zusammengesetzte (sekundäre) und benutzerdefinierte Varianten vor. Zentrale Bestandteile der ersten Gruppe sind dabei Operatoren auf Basis der relationalen Algebra sowie räumliche Operatoren. Sekundäre Operatoren stellen Funktionalität zur Auflösung objektivierter Relationen, zur Analyse zeitlicher, aufgabenbezogener Daten und zur Teilmodellextraktion sowie Modellreintegration bereit. Die sekundären Operatoren erleichtern damit eine Analyse und Weiterverarbeitung hochstrukturierter IFC-basierter Daten. Das Konzept benutzerdefinierter Operatoren wurde an zwei Beispielen erläutert. Der WallsInBasement-Operator selektiert Wände, die sich im Kellergeschoss des Gebäudes befinden. Der NextTaskWithLag-Operator wählt nachgelagerte, bauteilbezogene Aufgaben aus, wobei diese einen zeitlichen Abstand von mindestens einem Tag zur vorhergehenden Aufgabe aufweisen.

Als imperative Sprache stellt die Zuweisung von Ergebnissen an Variablen das zentrale Element der Programmierung in QL4BIM dar. Eine Anfrage kann somit als eine sequenzielle Folge derartiger Zuweisungen angesehen werden. Auf Grund dieser Anfragenstruktur ist eine komplette und eine schrittweise Ausführung einer Anfrage möglich. In der schrittweisen Ausführung können dem Endnutzer Zwischenergebnisse präsentiert werden, wodurch der Einfluss jedes Operators auf das Anfrageergebnis ersichtlich wird.

Die visuelle Notation ^{v}QL4BIM überführt die Anfragesprache in den Bereich der visuellen Programmierung. Dadurch kann der Benutzer durch graphische Repräsentationen in der Anfrageerstellung unterstützt werden. Die Eigenschaften der textuellen Notation bleiben dabei größtenteils erhalten und ^{v}QL4BIM stellt eine imperative Sprache dar, die auf der

Verknüpfung von Datenquellen, Datensenken und Operatoren beruht. Im Gegensatz zu datenflussorientierten Systemen bleibt die strikte Reihenfolge von Anweisungen bestehen. Operationen werden daher nicht bei Datenverfügbarkeit, sondern sequenziell ausgeführt.

Kapitel 7

Konzeption eines Laufzeitsystems für QL4BIM

Im vorigen Kapitel wurde QL4BIM als formale BIM-Anfragesprache vorgestellt. Zur Ausführung von QL4BIM-Anfragen bedarf es einer technischen Infrastruktur. Diese wird in diesem Kapitel konzipiert. Das beschriebene Laufzeitsystem hält die aktuelle Datenbasis vor, stellt Operatorenimplementierungen bereit und ermöglicht die Interpretation von Anfragen. Außerdem synchronisiert das System fortwährend beiden Anfragenotationen.

Als interaktive Analyseumgebung werden an das System hohe Anforderungen an die Ausführungsgeschwindigkeit von Anfragen gestellt. Dazu wurden in Kapitel 5 Algorithmen für die rechenintensiven räumlichen Operatoren diskutiert. Diese und alle weiteren, entwickelten Operatoren sind Bestandteil des Laufzeitsystems. Die beispielhaften Anfragen des letzten Kapitels zeigten außerdem, dass Relationen zwischen Entitäten häufig aufgelöst werden müssen. Ziel des präsentierten Konzepts ist somit auch die effiziente Dereferenzierung von Entitäten.

7.1 Überblick über die Komponenten des Laufzeitsystems

Das im Folgenden vorgestellte *Laufzeitsystem* (engl. *runtime system*, *RTS*) für QL4BIM kann in einer generellen, vorzugsweise objektorientierten Programmiersprache wie C++, C# oder Java umgesetzt werden. Abbildung 7.1 zeigt die Komponenten des Systems als UML-Klassendiagramm.

Der **Instance Parser** überführt Entitäten eines externen Schemas von einer serialisierten in eine *In Memory*-Repräsentation. Um die in Instanzdateien hinterlegten Daten mit schemabezogene Informationen anzureichern, nutzt der Instance Parser einen **Schema Parser**. Der **Data Pool** beinhaltet die In Memory-Repräsentationen der zu untersuchenden Entitä-

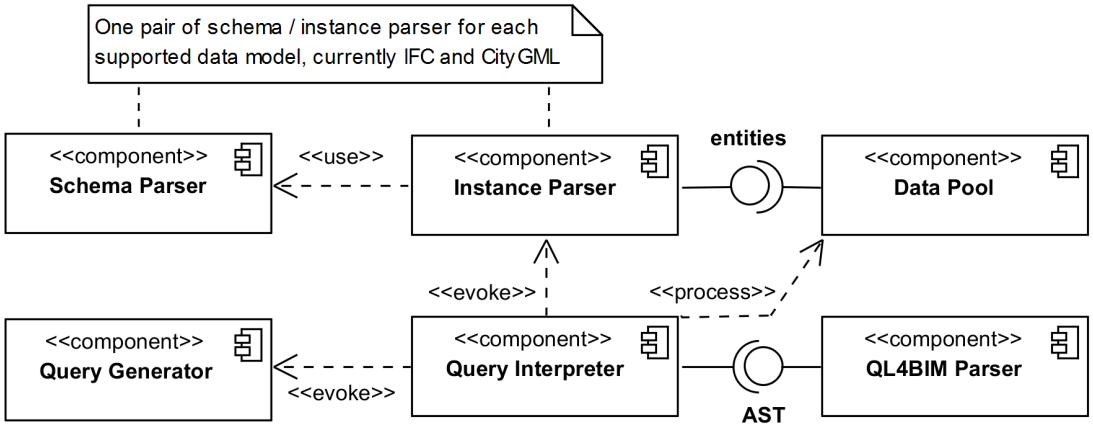


Abbildung 7.1: Die Komponenten des Laufzeitsystems von QL4BIM (UML-Klassendiagramm)

ten und ordnet diese mengenbasierten und relationalen Variablen zu. Der **QL4BIM Parser** überführt Anfragen der textuellen und visuellen Notation in eine geeignete *Zwischenrepräsentation* (engl. *intermediate representation*, *IR*). Die Hauptaufgabe des **Query Interpreter** ist die Filterung der Datenbasis durch die Verarbeitung dieser Zwischenrepräsentation. Der Interpreter ruft dazu die in ihm gekapselten Operatorimplementierungen auf. Neben der Datenfilterung führt der Interpreter auch fortlaufend die Synchronisation der textuellen und visuellen Anfragenotationen durch. Hierfür wird der **Query Generator** aufgerufen, der aus der Zwischenrepräsentation, einem *Abstract Syntax Tree*, Anfragen sowohl in ^{t}QL4BIM also auch in ^{v}QL4BIM generieren kann. Im Folgenden wird auf die RTS-Komponenten näher eingegangen und entsprechende Designentscheidungen erläutert.

7.2 Instance und Schema Parser

Ein Instanz-Parser und ein Schema-Parser müssen für jedes zu unterstützende Datenmodell in das RTS integriert werden. Für IFC-Instanzdateien wurde eine prototypische Implementierung auf Basis eines *Parser-Generators* umgesetzt (Wöß et al., 2003). Die entsprechende Grammatik ist im Anhang in Auflistung A.6 abgebildet. Der Parser für das EXPRESS-Schema der IFC kann auf die gleiche Art umgesetzt werden. Im Falle von CityGML liegen sowohl Instanz- als auch Schema-Daten im XML-Format vor. Dies erleichtert die Erstellung der beiden CityGML-Parser, da die XML-Unterstützung der verwendeten *Host*-Sprache eingesetzt werden kann (Stadler et al., 2009, Kap. 5).

Die jeweilige Kombination aus Instance Parser und Schema Parser ermöglicht die Vorhaltung auswertbarer Entitätsrepräsentation im Data Pool des Laufzeitsystems.

7.3 Data Pool

Generell bestehen zwei Möglichkeiten, die geparsten Entitäten innerhalb des Ablaufsystems vorzuhalten. In einem *Early Binding*-Ansatz müssen Klassendefinitionen für alle verwendeten Typen des externen Schemas in der Host-Sprache erstellt werden. Von diesen Klassen werden bei Bedarf während der Programmlaufzeit konkrete Instanzen erstellt. Beim *Late Binding* kann ein einzelner Datentyp genutzt werden, der Klassen- und Attributinformationen sowie konkrete Werte dynamisch aufnimmt und aggregiert (Zhou et al., 2014).

Für die Verarbeitung von IFC-Daten wurden bereits beide Ansätze eingesetzt. Strategien für ein Early Binding des IFC-Schemas werden in Nour & Beucke (2008) und Mazairac & Beetz (2013) aufgezeigt. Das in der *ISO 10303-22* spezifizierte *Standard Data Access Interface (SDAI)* stellt eine Programmierschnittstelle zur Verarbeitung von EXPRESS-basierten Daten bereit. Die Java-Implementierung der SDAI, als *JSDAI* bezeichnet, unterstützt sowohl Early Binding als auch Late Binding (Klein, 2010). Im Folgenden werden die beiden Bindungsvarianten für den Einsatz in einer Abfragesprache verglichen.

Die Prüfung der Typzugehörigkeit, in Hinblick auf eine Vererbungshierarchie, wird in einem Early Binding-Ansatz durch die in der objektorientierten Host-Sprache verfügbaren Funktionen ermöglicht. Dagegen ist der Zugriff auf Entitätsattribute über deren Bezeichner nur über das spezielle Sprachfeature *Reflection* möglich. Dieses beschreibt die Fähigkeit eines Programms, seinen Aufbau zu untersuchen und zu verändern (Maes, 1987). Die Unterstützung von Reflections variiert unter den gängigen Host-Sprachen. Der massive Einsatz des Sprachfeatures kann sich zudem negativ auf die Performance auswirken (Wienholt, 2004, S. 232). Zusätzlich zur Generierung der Klassendefinitionen, erfordert diese Einschränkung des Early Bindings eine Zuordnung von Attributsbezeichnungen mit Attributsfunktionen. Beispielsweise kann somit durch Übergabe der Zeichenkette `OwnerHistory` das Enitätsattribut *OwnerHistory* ausgewertet werden.

Beim Einsatz von Late Binding werden Typauswertungen nicht unterstützt. Attribute lassen sich über ihren Bezeichner jedoch direkt auswerten. Die Typauswertung kann durch den erweiterten Einsatz des Schema-Parsers ausgeglichen werden. Dieser erstellt dazu eine Vererbungshierarchie in Form einer Baumstruktur, die zur Programmlaufzeit ausgewertet werden kann. Tabelle 7.1 fasst die Vor- und Nachteile beider Bindungsvarianten zusammen.

Im Falle des Ablaufsystems für eine BIM-Anfragesprache steht die Auswertbarkeit von Entitäten bezüglich ihrer Typisierung und ihrer Attributwerte in Vordergrund. Die zwei Bindungsvarianten unterstützten derartige Auswertungen jeweils in einem Bereich. Spätes Binden hat den wichtigen Vorteil, dass die Kopplung zu externen Schemata geringer ausfällt und zusätzliche Objektmodelle mit geringerem Implementierungs-

Bindung	Typauswertung: entity is „IfcWall“	Attributauswertung: entity.GetValue(„Name“)	Benötigte Erweiterung für die Datenanalyse
Early Binding	Unterstützt	Nicht unterstützt	Verknüpfung von Attributbezeichnungen zu Eigenschaften
Late Binding	Nicht unterstützt	Unterstützt	Vererbungshierarchie als Baumstruktur

Tabelle 7.1: Unterstützung der Typ- und Attributauswertung von Entitäten nach Bindungsvariante

aufwand unterstützt werden können. Daher wird für das QL4BIM-Laufzeitsystem Late Binding genutzt.

Als zentrale Entitätsrepräsentation des genutzten Late Bindings dient die bereits vorgestellte ExternalEntity-Klasse. Abbildung 7.2 zeigt deren Aufbau als UML-Klassendiagramm. Eine ExternalEntity setzt sich aus EntityParts zusammen die jeweils ein einzelnes Entitätsattribut kapseln. Über die Methoden `GetName`, `GetType` und `GetValue` können der Bezeichner, die Typisierung und der Wert des Attributs ermittelt werden. In den Operatorimplementierungen werden diese Methoden genutzt um Attribute auszuwerten. Über das Attribut *ClassNumber* der ExternalEntity kann über eine Integer-Repräsentation die Klassenzugehörigkeit der gekapselten Entität effizient geprüft werden.

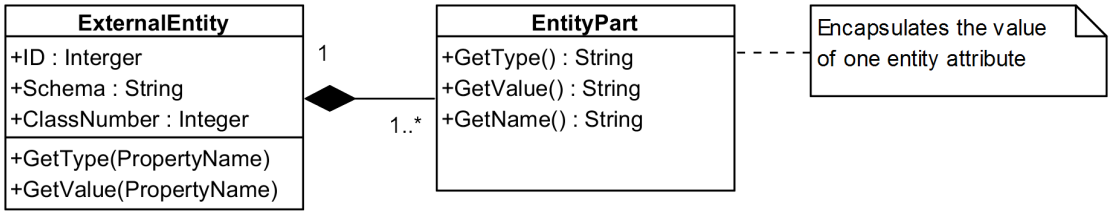


Abbildung 7.2: Die ExternalEntity-Klasse ist zentrales Element des Late Binding-Ansatzes und besteht aus dynamisch erstellten EntityParts (UML-Klassendiagramm)

Neben der Vorhaltung auswertbarer Entitätsrepräsentationen ist der Data Pool für die effiziente Selektion und Dereferenzierung von Entitäten konzipiert. Die Auswahl von Entitäten und die Auflösung von Referenzbeziehungen werden im Laufzeitsystems durch eine *HashTable*-Struktur realisiert (Shaffer, 2012, Kap. 9.4). Als Schlüssel dient ein stets eindeutiger, durch das Laufzeitsystem verwalteter Entitätsbezeichner. Abbildung 7.3 stellt Referenzen zwischen feingranularen IFC-Entitäten dar und verdeutlicht den direkten Datenzugriff auf diese mittels der HashTable-Struktur.

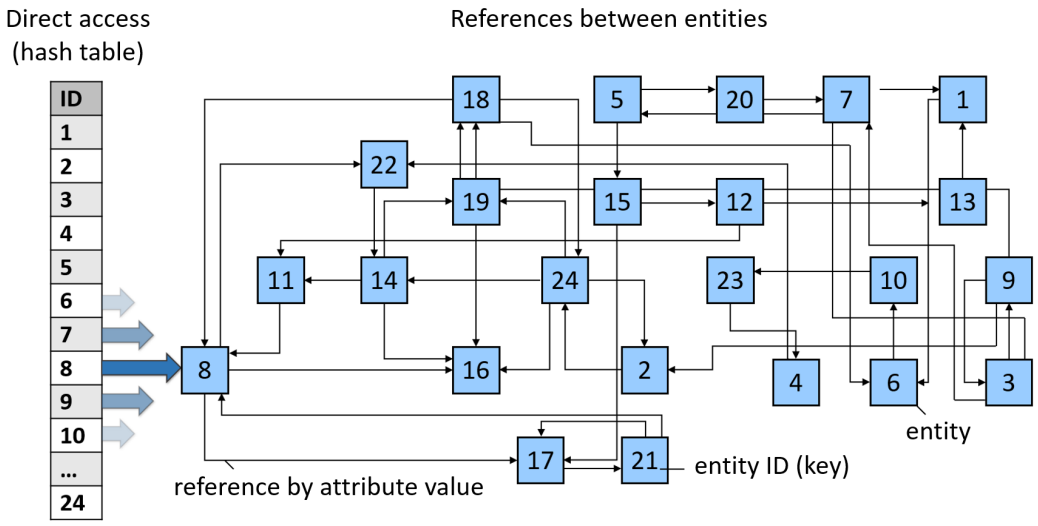


Abbildung 7.3: HashTable-Struktur zur Auflösung von Referenzbeziehungen zwischen Entitäten

Der HashTable ermöglicht die Selektion eines Elements auf Basis eines Schlüssels in konstanter Zeitkomplexität, also unabhängig von der Anzahl an Elementen innerhalb der Struktur. Dies wirkt sich positiv auf die Ausführungsgeschwindigkeit von Anfragen aus in denen feingranulare Entitäten über Attribute zu weiteren Entitäten verweisen. Daraus ergeben sich Graphstrukturen, die bei Analysen wiederholt traversiert und ausgewertet werden müssen. Die eingesetzte Referenzierungsmethode ist somit speziell für die Verarbeitung von hochstrukturierten IFC-Daten ausgelegt.

7.4 QL4BIM Parser und AST-Erstellung

QL4BIM-Anfragen sind Sätze die aus dem visuellen und textuellen Alphabet der Anfragesprache gebildet werden. Zur Verarbeitung derartiger sprachbasierter Repräsentationen hat sich ein *Abstract Syntax Tree (AST)* als geeignete Zwischenrepräsentation etabliert (Louden & Lambert, 2012, S. 216). Die Baumstruktur bildet die Sprachelemente in ihrem hierarchischen und sequenziellen Aufbau ab und ermöglicht so die syntaktische Interpretation von Anfragen.

Innerhalb des Laufzeitsystems müssen sowohl ^{t}QL4BIM als auch ^{v}QL4BIM durch den Anfrageparser in einen AST überführt werden. Während die Verarbeitung der textuellen Notation auf der Erkennung grammatikalischer Elemente basiert, wird in der graphischen Variante die gerichtete Graphstruktur der visuellen Elemente in den AST transformiert.

Zur Generierung eines ASTs aus einer ^{t}QL4BIM-Anfrage müssen die relevanten, semantischen Elemente identifiziert und syntaktische Auszeichnungen entfernt werden. Der reduzierte

Syntaxbaum erlaubt dadurch ein einfachere Verarbeitung als ein *Parse Tree*, der alle Sprachelemente enthält (Louden & Lambert, 2012, S. 213). ASTs existieren in unterschiedlichen Varianten. Im Laufzeitsystem von QL4BIM wird ein *Irregular Heterogeneous AST* eingesetzt (Parr, 2010, S. 114). Dieser Baum besteht aus Knoten mit unterschiedlicher Typisierung. Kinderknoten werden durch Attribute in ihren Elternknoten referenziert.

In QL4BIM wird der AST in der Anfrageausführung genutzt. Außerdem ermöglicht die Struktur eine vorgelagerte, globale Anfrageüberprüfung und dient zur Synchronisation beider QL4BIM-Notationen. Wegen der geringen Anzahl an Sprachkonstrukten lässt sich ein *Irregular Heterogeneous AST* mit geringen Aufwand erstellen und bietet eine hochstrukturierte Zwischenrepräsentation von Anfragen.

Im folgenden wird die Erstellung des AST anhand einer textuellen Anfrage diskutiert. Auflistung 7.1 enthält die ersten beiden Aussagen von Musteranfrage 2 (Wand/Tür-Paare). Abbildung 7.4 zeigt die Überführung dieser Aussagen in den entsprechenden AST.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|opening] = Deassociater(entities.HasOpenings)
```

Auflistung 7.1: Die ersten beiden Aussagen von Musteranfrage 2 in ᵗQL4BIM

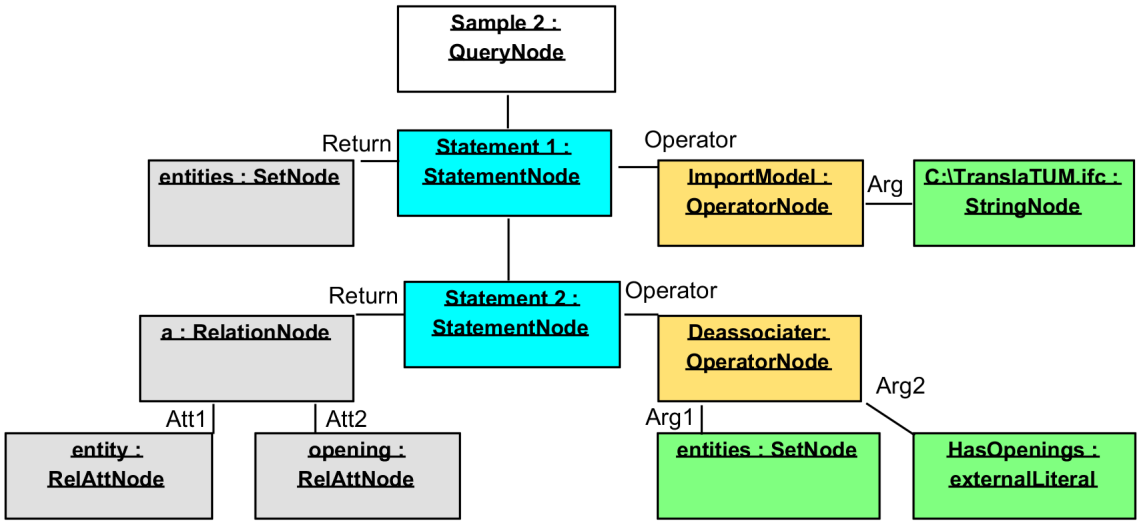


Abbildung 7.4: AST der ersten beiden Aussagen von Musteranfrage 2

Der QL4BIM-AST besteht aus einer QueryNode als Wurzelknoten. An dieser werden StatementNodes in geordneter Form angefügt. Jede StatementNode referenziert Knoten für die Repräsentation der zu belegenden Variable und des genutzten Operators. Im Fall einer mengenbasierten Variablenzuweisung reicht ein einzelner SetNode-Knoten. Bei der Verwendung

einer relationalen Variable wird ein übergeordneter Knoten vom Typ RelationNode generiert. Die relationalen Attribute werden als RelAttNodes vorgehalten. Des Weiteren beschreiben die Kindknoten von Operatoren die jeweils übergebenen Parameter, wie Variablen, Prädikate und einfache Konstanten.

7.5 Query Interpreter

Der Query Interpreter verarbeitet den vom Anfrageparser bereitgestellten AST. Dazu beinhaltet die Komponente mehrere *Visitatoren*. Ein Visitator ermöglicht es, Operatoren auf einer Datenstruktur auszuführen, ohne die Elemente der Struktur anpassen zu müssen.

> Der Einsatz des Visitor-Entwurfsmuster wird empfohlen, wenn durch Verarbeitung einer stabilen Datenstruktur unterschiedliche Aktionen ausgeführt werden sollen (Gamma et al., 1994, Kap. 2.8). Somit bietet sich dieses Muster für die AST-Verarbeitung in QL4BIM an, da hier Anfragen validiert, ausgeführt und in unterschiedliche Notation überführt werden müssen.

Im QL4BIM-RTS kommen drei Visitatoren zum Einsatz. Der **Symbol-Visitor** erkennt die unterschiedlichen Symbole der Sprache und prüft deren semantische Verwendung. Variablen werden außerdem durch diesen Visitator in eine Symboltabelle eingefügt. Auf Grund der Regeln zum Geltungsbereich von Variablen in QL4BIM besitzt der globale Bereich an Aussagen und jeder benutzerdefinierte Operator eine eigene Symboltabelle.

Der **Operator-Visitor** überprüft, ob der genannte Operator im RTS vorhanden ist. Außerdem wird durch den Visitor sichergestellt, dass eine passende Operatorüberladung für die jeweils übergebenen Parametersymbole vorhanden ist. Standardmäßig wird dabei die Anzahl und Typisierung von Argumenten sowie der verwendete Rückgabetyp überprüft. Je nach Operator müssen mitunter komplexere Regeln validiert werden. Im *Projector*-Operator muss beispielsweise die Anzahl der Attribute in der Rückgaberelation mit der Anzahl an attributiven Parametern übereinstimmen (siehe Auflistung 7.2, Z. 4).

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|open] = Deassociater(entities.HasOpenings)
3 b[entity|open|filling] = Deassociater(a, [open].HasFilling)
4 c[entity|filling] = Projector(b, [entity], [filling])
```

Auflistung 7.2: Der Operator-Visitor vergleicht die Anzahl der übergebenen, relationalen Attribute an den Projector-Operator mit der Stelligkeit der relationalen Variable

War die semantische Anfragevalidierung durch die beiden Visitatoren erfolgreich, verarbeitet der **Execution-Visitor** den AST. Dabei wird für jeden Operator die entsprechende Imple-

mentierung mit den aktuellen Parametern aufgerufen. Das jeweilige Operationsergebnis wird unter der Bezeichnung der Rückgabevariable im Data Pool in der aktuellen Symboltabelle abgelegt.

Die aktuelle Symboltabelle befindet sich stets an oberster Position eines *Funktionsstapels*. Der Stapel wächst, wenn benutzerdefinierter Operatoren aufgerufen werden und schrumpft wenn derartige Operatoren verlassen werden. Beim Verlassen des Operators und dem damit verbundenen Abräumen der aktuellen Symboltabelle wird die Rückgabevariable in die darunterliegende Symboltabelle übertragen. Dies ist die standardmäßige Implementierung für Sprachen mit *funktionsbasierten Geltungsbereichen* (Parr, 2010, Kap. 6).

> Die Struktur des ASTs in QL4BIM ermöglicht zwei Ausführungsvarianten für valide Anfragen. So kann eine Anfrage durch den Execution-Visitor komplett ausgeführt werden. Dem Endnutzer präsentiert sich dann das finale Ergebnis der Anfrage. Die Alternative dazu ist eine schrittweise Ausführung. In dieser wird nach der erfolgreichen Prüfung des kompletten ASTs jede Aussage auf Benutzeraktion hin ausgeführt. Angezeigt wird stets die in der Aussage verwendete Variable. Dies ist vergleichbar mit der Debugging-Funktionalität in allgemeinen Programmiersprachen. Durch den Aufbau von Anfragen lässt sich derartige Funktionalität direkt in das Ausführungsmodell von QL4BIM integrieren.

Der Query Interpreter steuert somit die Überprüfung und Ausführung von Anfragen. Des Weiteren stellt der Interpreter die elementaren und sekundären Operatoren der Anfragesprache als Methoden zur Verfügung. Am Beispiel des `TypeFilter`s wird das Vorgehen zur Realisierung eines Operators verdeutlicht. Auflistung 7.3 zeigt die Umsetzung des Operators in seiner ersten, mengenbezogenen Überladung. Der Interpreter übergibt zwei SetNodes und eine ExternalIdentifierNode aus dem AST an die Filtermethode.

```
1 void TypeFilter(SetNode returnSet, SetNode inputSet, ExIdentifierNode type)
2     var tempReturnSet = new SetNode()
3     allTypes = IfcSchemaParser.GetAllSubtypes(type)
4     foreach entity in inputSet.Values
5         if allTypes.Contains(entity.Type)
6             tempReturnSet.Values.Add(entity)
7     returnSet = tempReturnSet
```

Auflistung 7.3: Pseudo-Code des TypeFilters in Laufzeitsystem der Anfragsprache

Über den IfcSchemaParser können alle Subtypen des angegebenen Typs bestimmt werden (Z. 3). Daraufhin wird jede in der Eingangsmenge enthaltene Entität auf Übereinstimmung mit einem dieser Typen hin überprüft (Z. 4 u. 5). Tritt eine Übereinstimmung auf, wird die Entität einer temporären Rückgabevariable hinzugefügt (Z. 6). Eine temporäre Rückgabeva-

riable wird benötigt, da QL4BIM die gleichzeitige Verwendung eines Symbols als Parameter und Rückgabevariable unterstützt. `entities = TypeFilter(entities is IfcWall)` ist ein derartiges Statement. Somit können sich ein Parameter und eine Rückgabevariable auf das identische Symbol beziehen. Die temporäre Variable wird daher erst im letzten Schritt eines Operators der eigentlichen Rückgabevariable zugewiesen (Z. 7).

7.6 Query Generator

Eine Besonderheit von QL4BIM ist die Unterstützung zweier Notationen. Dient der AST primär zur Anfragevalidierung und -ausführung, kann dieser auch genutzt werden, um die gleichzeitige Verwendung beider Notationen zu ermöglichen. Daher wird diese Baumstruktur bei jeder Änderung der aktuellen Anfrage aktualisiert, auch wenn keine Anfragenausführung durch den Endnutzer gestartet wird. Der Query Generator verarbeitet anschließend den AST und erstellt daraus eine neue Anfrage. Diese Generierung kann in zwei Richtungen ablaufen. Bei einer Änderung in der textuellen Sprache erzeugt der Query Generator eine visuelle Anfrage und umgekehrt. Die sich daraus ergebende fortlaufende Synchronisation der Notationen wird in Abbildung 7.5 veranschaulicht.

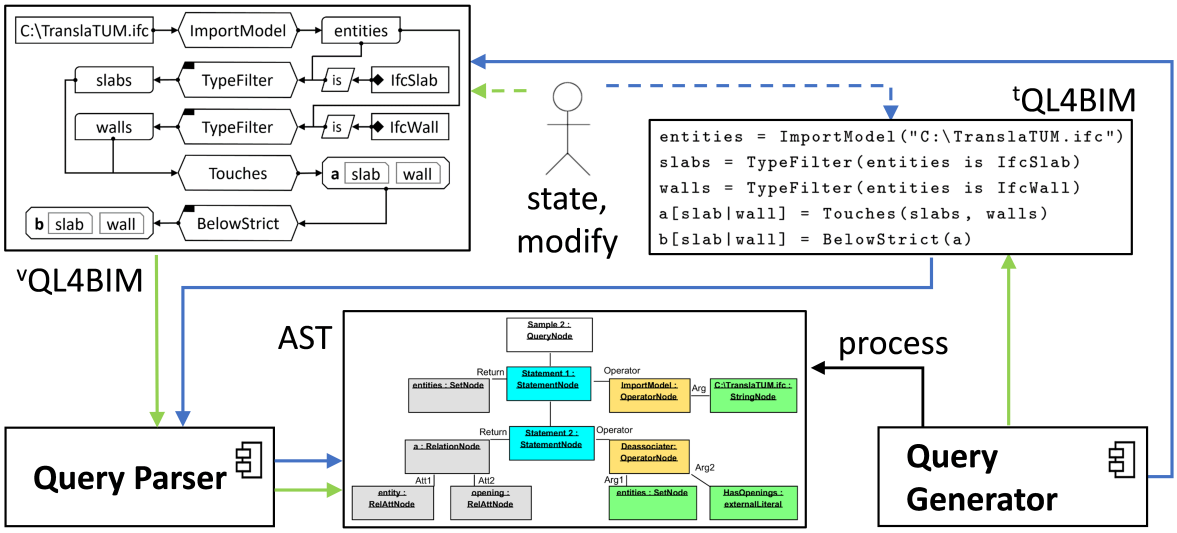


Abbildung 7.5: Synchronisation der beiden Sprachnotationen durch den Query Parser und den Query Generator.

7.7 Exception Handling

Bei der Ausführung einer QL4BIM-Anfrage können *Ausnahmen* (engl. *exceptions*) auftreten. Beispielsweise kann ein Operator mit Parametern aufgerufen werden, die zu keiner verfügbaren Überladung passen. Einige Ausnahmen wie beispielsweise `UnknownSignature` verhindern

eine Ausführung einer Anfrage. Andere Ausnahmearten wie `EmptyRelation` können als Warnhinweise interpretiert werden. Tabelle 7.2 beinhaltet die unterschiedlichen Ausnahmearten der Anfragesprache.

QL4BIM-Excetion	Beschreibung
EmptySet	Einer mengenbasierten Variable wird eine leere Menge zugewiesen
EmptyRelation	Einer relationalen Variable wird eine leere Relation zugewiesen
UnknownIndex	Das relationale Attribut ist in der Relation nicht vorhanden
IndexOutOfBound	Der Index überschreitet die Attributanzahl der Relation
DuplicatedIdentifier	Ein Bezeichner wird nicht regelkonform, mehrfach verwendet
UnknownSignature	Es existiert keine passende Signatur des Operators
UnknownOperator	Kein Operator mit dieser Bezeichnung vorhanden
UnknownVariable	Keine Variable mit dieser Bezeichnung im Geltungsbereich
UnknownAttribute	Kein relationales Attribut mit der Bezeichnung im Geltungsbereich
UnknownSchema	Das referenzierte Schema wird nicht unterstützt
DuplicatedOperator	Der Operator ist mit identischer Signatur bereits vorhanden
UnknownType	Der Typ ist im genutzten Schema nicht vorhanden
UnknownAttribute	Das Entitätsattribut ist im genutzten Schema nicht vorhanden
RelArityMismatch	Die relationale Variable besitzt eine unpassende Stelligkeit

Tabelle 7.2: Auflistung und Erläuterung aller QL4BIM-Exceptions

7.8 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie ein Laufzeitsystem für QL4BIM strukturiert und implementiert werden kann. Dies umfasst die Komponenten zur Verarbeitung der textuellen und visuellen Anfragen und deren Synchronisierung. Außerdem wurde auf die Vorhaltung externer Entitäten im Datenpool des Analysesystems eingegangen und ein Ansatz zur effizienten Auflösung von Referenzbeziehungen beschrieben. Eine weitere zentrale Komponente des Laufzeitsystems stellt der Query Interpreter dar. Die Komponente verarbeitet Anfragen auf Basis der entwickelten, textuellen und visuellen Grammatik und stellt alle Operatoren der Sprache bereit.

Neben der Unterstützung zweier, synchronisierter Notationen besitzt das Laufzeitsystem eine weitere Besonderheit. Die Anfragenstruktur in QL4BIM erlaubt eine komplette und eine schrittweise Ausführung durch den Sprachinterpreter. In der schrittweisen Ausführung ist die Auswirkung jeder einzelnen Aussage für den Endnutzer nachvollziehbar. Es wird erhofft, dass hierdurch Anwender in die Lage versetzt werden, komplexere Anfragen zu erstellen. Auf Basis der beschriebenen Systemarchitektur konnte QL4BIM prototypisch

implementiert werden. Wie im nachfolgenden Kapitel ersichtlich, ermöglicht dies eine ausführliche Validierung der Forschungsansätze durch empirische Experimente.

Kapitel 8

Validierung von QL4BIM

In diesem Kapitel wird die Einsatzeignung von QL4BIM für die Analyse, Überprüfung und Weiterverarbeitung von BIM-Daten diskutiert. Die Validierung gliedert sich in vier Teilbereiche. Zum einen wird der durch QL4BIM abgedeckte Funktionsumfang aufgezeigt und bewertet. Danach wird die Laufzeitumgebung der Sprache untersucht. Diese wurde auf Basis der im vorhergehenden Kapitel beschriebenen Systemarchitektur umgesetzt. Mit Hilfe dieser prototypischen Implementierung können die Ausführungszeiten von Analysen gemessen und bewertet werden. Im dritten Abschnitt dieses Kapitels wird der Umfang der Sprachgrammatik von QL4BIM und generellen Anfragesprachen verglichen. Den letzten Teil der Validierung stellt eine Benutzerbefragung dar. In dieser wurden Fachanwender zur Verständlichkeit und Einsetzbarkeit von QL4BIM befragt.

8.1 Betrachtung des sich ergebenden Funktionsumfangs

Der folgende Abschnitt beschreibt den Funktionsumfang der Sprache und der Ablaufumgebung in den Bereichen 4D-Anfragen, Konsistenzüberprüfung, Teilmodellerstellung und -Reintegration sowie der kombinierten Analyse von Gebäude- und Stadtmodellen.

8.1.1 4D-Anfragen

Die Anfrage in Auflistung 8.1 illustriert die Unterstützung von QL4BIM hinsichtlich 4D-Analysen. Zu Beginn werden die Wände des zweiten Stockwerks selektiert (Z. 2 - 5). Danach werden von diesen Wänden die Aufgaben dereferenziert, die sich auf die Schalungserstellung beziehen und am 01. Januar 2015 beginnen (Z. 6 - 7). Im Folgenden werden die schalungsbezogenen Aufgaben von Platten selektiert (Z. 8 - 9). In Zeile 10 werden dem zeitlichen Operator `Overlays` die Wand-Aufgaben- und die Platten-Aufgaben-Relationen übergeben und nach

sich zeitlich überlagernden Aufgaben gesucht. Nach einer Projektion liefert die Anfrage eine Relation mit Stelligkeit 4 zurück. Deren Tupeln bestehen aus einer Wand des zweiten Stocks, wandbezogener Schalungsaufgabe, Platte und plattenbezogener Schalungsaufgabe. Die beiden Aufgaben überlagern sich zeitlich und die Aufgabe der Wand beginnt am 01. Januar 2015.

```
1  entities = ImportModel("C:\TranslaTUM.ifc")
2  walls = TypeFilter(entities is IfcWall)
3  a[wall|storey] = Deassociater(walls.ContainedInSpatialStructure)
4  a[wall|storey] = AttributeFilter(a, [storey].Name = "SecondFloor")
5  walls = Projector(a, [wall])
6  b[wall|task|time] = TimeResolver(walls,"formwork")
7  b[wall|task|time] = AttributeFilter(b, [time].ScheduleStart = 01-01-2015)
8  slabs = TypeFilter(entities is IfcSlabs)
9  c[slab|task|time] = TimeResolver(slabs,"formwork")
10 d[wall|wTask|wTime|slab|sTask|sTime] =  Overlays(b, b[time], c, c[time])
11 c[wall|wTask|slab|sTask] = Projector(c,[1], [2], [4], [5])
```

Auflistung 8.1: Beispielhafte 4D-Anfrage in QL4BIM, räumliche Analyse über Relationsauswertung

In der vorherigen Anfrage wurde die räumliche Auswertung auf Basis der expliziten Verknüpfung zwischen Bauteil und Stockwerk realisiert. Die Anfrage in Auflistung 8.2 nutzt hingegen den räumlichen Operatoren Touches in Kombination mit dem zeitlichen Operator Identical um eine 4D-Analyse umzusetzen. Die Anfrage beginnt mit einer Typ- und Attributfilterung. Somit wird eine Menge aus Kanälen und eine diesbezügliche Teilmenge gewonnen (Z. 2 - 3). In den Zeilen 4 und 5 werden die jeweiligen Aufgaben zu den Kanälen dereferenziert. Der Identical-Operator findet die Kanäle, die jeweils im identischen Zeitintervall installiert werden (Z. 6). Nach der Entfernung von nicht mehr benötigten zeitlichen Entitäten in Zeile 7 werden Kanalpaare auf Berührung geprüft. Somit erhält man eine Relation aus Kanalpaaren. Der erste Kanal besitzt einen bestimmten Wert im *Name*-Attribut, berührt den zweiten und beide Bauteile werden im gleichen Zeitintervall installiert.

```
1  entities = ImportModel("C:\TranslaTUM.ifc")
2  ducts = TypeFilter(entities is IfcDuct)
3  someDucts = AttributeFilter(ducts.Name = "duct#101")
4  a[main|task|time] = TimeResolver(someDucts)
5  b[parts|task|time] = TimeResolver(ducts)
6  d[] = Identical(a, a[time], b, b[time])
7  d[] = Projector(c,[1], [4])
8  d[main|part] = Touches(d, 10)
```

Auflistung 8.2: Beispielhafte 4D-Anfrage in QL4BIM mit räumlicher Analyse über den Touches-Operator

Die gezeigten Anfragen illustrieren die Möglichkeiten mit QL4BIM 4D-Analysen auszuführen. Dabei können die räumlichen Aspekte durch im Modell explizit vorgehaltene Relationen ausgewertet werden. Als Alternative können die räumlichen Entitätsrepräsentationen in vielfacher Weise verarbeitet werden. Durch die Kombination von räumlichen, zeitlichen und semantischen Operatoren der Anfragesprache lassen sich detaillierte 4D-Auswertungen erstellen, die mit anderen Ansätzen auf dem gezeigten Abstraktionsniveau nicht realisierbar sind.

8.1.2 Überprüfung der Modellkonsistenz

In einem IFC-Modell können semantische, geometrische und zeitliche Informationen vorgehalten werden. Teilweise überschneiden sich die Daten dieser Bereiche. Konzeptionell lassen sich diese Redundanzen durch die unterschiedlichen Modellierungstechniken rechtfertigen. Dadurch liegen die Redundanzen implizit vor und sind nur nach einer Prozessierung der Daten erkennbar.

Die räumlich-semantische Konsistenz von Gebäudemodellen ist gegeben, wenn die semantische Modellierung räumlicher Gegebenheiten mit den ebenfalls vorhandenen geometrischen Daten übereinstimmt. Diesbezügliche Redundanzen existieren im IFC-Format mitunter bei der Modellierung räumlicher Strukturen und der Zuordnung von Bauteilen zu diesen Strukturen. Auflistung 8.3 zeigt diesbezüglich die Unterklassen von IfcSpatialStructureElement.

```
1 ENTITY IfcSpatialStructureElement
2 ABSTRACT SUPERTYPE OF(ONEOF(IfcBuilding,IfcBuildingStorey,IfcSite,IfcSpace))
3 SUBTYPE OF IfcSpatialElement; ...
4 END_ENTITY;
```

Auflistung 8.3: Unterklassen von IfcSpatialStructureElement, aus IFC-Schema Version 4

Räumliche Strukturen wie IfcSite und IfcBuilding bilden im IFC-Schema sowohl eine semantische als auch eine räumliche Hierarchie. IfcSite stellt dabei das Wurzelobjekt der Modellierung dar. Zur Verknüpfung der räumlichen Strukturen dient das objektivierte Relationsobjekt IfcRelAggregates. Abbildung 8.1 zeigt, dass Bauteile wie Wände und Stützen über IfcRelContainedInSpatialStructure ihren beherbergenden Stockwerken zugeordnet werden können.

Die folgende Analyse räumlich-semantischer Konsistenz mit QL4BIM untersucht im ersten Schritt die räumliche Konfiguration zwischen semantisch verknüpften IfcSites, IfcBuildings, IfcBuildingStoreys und IfcSpaces. In zwei weiteren Anfragen werden die topologischen Beziehungen zwischen Bauteilen und ihren übergeordneten räumlichen Strukturen überprüft.

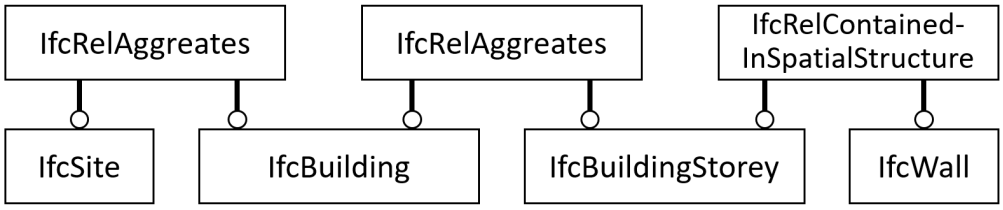


Abbildung 8.1: Verknüpfung hierarchischer, räumlicher Strukturen mit Bauteilen, EXPRESS-G-Diagramm

Auflistung 8.4 beinhaltet die erste Anfrage, in der nach dem Import des Modells eine Selektion aller IfcSpatialStructureElement-Instanzen ausgeführt wird. Der Deassociater-Operator navigiert über das inverse Attribut *IsDecomposedBy* zur IfcRelAggregates-Entität (Z. 3). Implizit dereferenziert der Operator zusätzlich das gegenläufige Attribut *RelatedObjects* des objektivierten Relationsobjekts. Nach einer Typfilterung in Zeile 4 beinhaltet die relationale Variable `b[site|building]` semantisch verknüpfte Paare aus IfcSite und IfcBuilding. In der folgenden topologischen Abfrage wird überprüft, ob sich alle Gebäude innerhalb der räumlichen Baustellenstruktur befinden (Z. 5). Durch die Verwendung des toleranzunterstützenden `Contains`-Operators kann dabei die Einhaltung eines benutzerdefinierten Abstands zur übergeordneten Struktur sichergestellt werden. Die gefundenen Paare werden von der ursprünglichen Auswahl entfernt (Z. 6). Die Relation `d[site|building]` beinhaltet zum Ende der Anfrage die zu überprüfenden Paare aus räumlichen Strukturen.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 spatialStructures = TypeFilter (entities  is IfcSpatialStructureElement)
3 a[structure|spatial] = Deassociater(spatialStructures.IsDecomposedBy)
4 b[site|building] = TypeFilter (a,[1] is IfcSite,[2] is IfcBuilding)
5 c[site|building] = Contains(b, 1.5)
6 d[site|building] = Difference(b, c)
```

Auflistung 8.4: Räumlich-semantische Konsistenzprüfung mit QL4BIM, Anfrage 1 (räumliche Strukturen)

In Abbildung 8.2 wird eine beispielhafte Konstellation zwischen einer IfcSite und drei referenzierten IfcBuildings dargestellt. Wird die vorherige Abfrage aus Auflistung 8.4 auf das entsprechende IFC-Modell angewendet, genügt nur `IfcBuilding #3` der Überprüfung. Das erste Gebäude befindet sich zwar komplett innerhalb der IfcSite-Geometrie jedoch nicht im geforderten Abstand zu dieser. Das zweite Gebäude schneidet die Baustellengeometrie.

Die zweite QL4BIM-Anfrage in Auflistung 8.5 überprüft das Enthaltensein von Bauteilen in räumlichen Strukturen. Die Anfrage ähnelt der Überprüfung hierarchischer räumlicher Strukturen. Hier wird jedoch über den `Deassociater`-Operator in Zeile 3 eine Relation zwischen Stockwerken und beherbergten Bauteilen aufgebaut. Durch den `Contains`-Operator

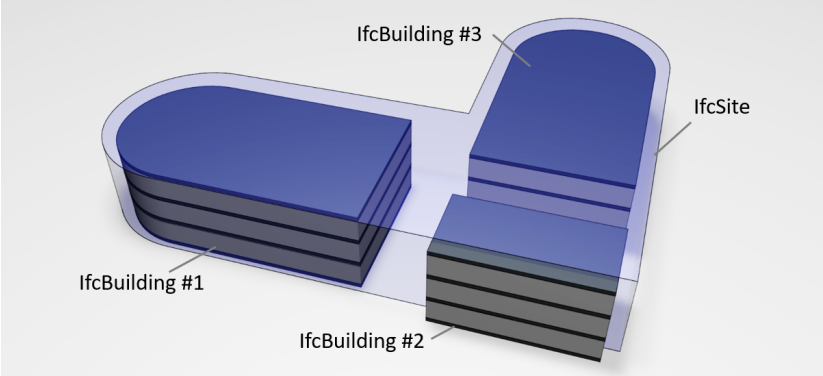


Abbildung 8.2: Veranschaulichung der räumlich-semantischen Konsistenzprüfung, Baustellengeometrie in Relation zu Gebäuden, Anfrage 1 (räumliche Strukturen)

wird überprüft, ob sich alle Bauteile innerhalb des referenzierten Stockwerks befinden (Z. 4). Durch abschließende Differenzbildung sind in Variable `c[storey|element]` diejenigen Paare vorhanden, für die die untersuchte räumlich-semantische Konsistenz nicht gegeben ist.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 stories = TypeFilter(entities is IfcBuildingStorey)
3 a[storey|element] = Deassociater(stories.ContainsElements)
4 b[storey|element] = Contains (a, 0.01)
5 c[storey|element] = Difference(a, b)
```

Auflistung 8.5: Räumlich-semantische Konsistenzprüfung mit QL4BIM, Anfrage 2 (Stockwerke und beherbergte Bauteile)

In Abbildung 8.3 wird eine beispielhafte Konstellation zwischen IfcBuildingStorey- und IfcColumn-Instanzen dargestellt. Wird die vorherige Abfrage aus Auflistung 8.5 auf das entsprechende IFC-Modell angewendet, genügen die markierten Stützen nicht der Überprüfung. Diese sind semantisch dem ersten Stockwerk zugeordnet, befinden sich jedoch geometrisch innerhalb des dritten Stockwerks.

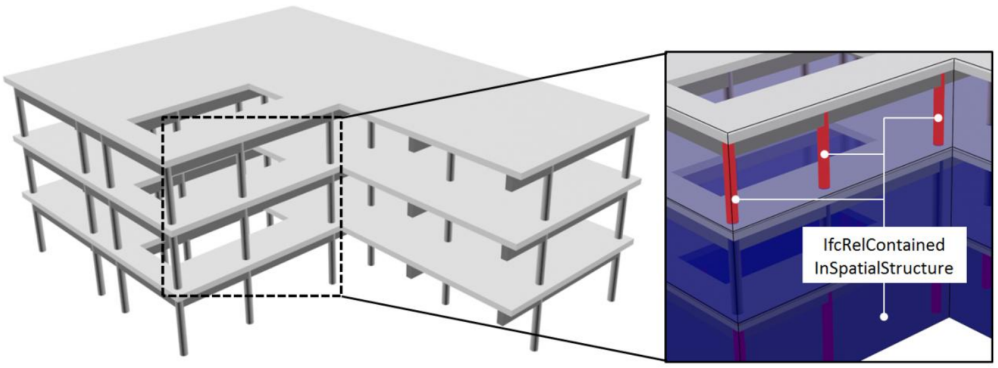


Abbildung 8.3: Veranschaulichung der räumlich-semantischen Konsistenzprüfung in Anfrage 2 (Stockwerke und beherbergte Bauteile)

Nicht immer ist eine 1:1-Beziehung zwischen Bauteilen und Stockwerken gegeben. Als Beispiel sei auf die Modellierung einer Vorhangfassade (IfcCurtainWall) verwiesen. Diese kann über die objektivierte Relation IfcRelContainedInSpatialStructure beispielsweise dem Erdgeschoss zugeordnet werden, obwohl die Fassadengeometrie über dieses Stockwerk herausragt. Zur Verknüpfung der verbleibenden Stockwerke mit der Fassade dient IfcRelReferencedInSpatialStructure.

In der nachfolgenden Anfrage in Auflistung 8.6 werden Paare aus Stockwerk und Bauteil gebildet. Dabei werden Verknüpfungen durch IfcRelContainedInSpatialStructure und IfcRelReferencedInSpatialStructure berücksichtigt (Z. 2 - 3). In Zeile 4 werden die gewonnenen Relationen vereinigt und in Zeile 5 auf die Typen IfcBuildingStorey und IfcCurtainWall eingegrenzt. Durch den `Overlaps`-Operator in Zeile 6 werden diejenigen Paare identifiziert, in denen die Fassadengeometrie mit dem referenzierten Stockwerk überlappt. Durch abschließende Differenzbildung sind in der Variable `f[storey|curWall]` diejenigen Paare vorhanden, für die die untersuchte räumlich-semantische Konsistenz nicht gegeben ist.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 a[entity|element] = Deassociater(entities.ReferencesElements)
3 b[entity|element] = Deassociater(entities.ContainsElements)
4 c[entity|element] = Union(a, b)
5 d[storey|element] = TypeFilter (c,
                      [1] is IfcBuildingStorey, [2] is IfcCurtainWall)
6 e[storey|curWall] = Overlaps (d)
7 f[storey|curWall] = Difference(d, e)
```

Auflistung 8.6: Räumlich-semantische Konsistenzprüfung mit QL4BIM, Anfrage 3 (Stockwerke und Vorhangfassaden)

Neben den betrachteten Fällen zur räumlich-semantischen Konsistenzprüfung mit QL4BIM sind weitere Anwendungsmöglichkeiten vorhanden. Tabelle 8.1 nennt diesbezüglich vier Modellierungsbereiche, die jeweils auszuwertende semantische Relation und die auszuführende räumliche Analyse.

Neben der Verarbeitung semantischer und räumlicher Daten kann QL4BIM genutzt werden, um raum-zeitlich-semantische Aspekte des Gebäudemodells zu analysieren. Damit können Plausibilitätsprüfungen der Terminplanung durchgeführt werden. Beispielsweise können die im Modell enthaltenen Bauteile und ihre aufgabenbezogenen zeitlichen Daten hinsichtlich technologischer Abhängigkeiten ausgewertet werden.

Das Vorgehen wird im Folgenden an Stützen und Bodenplatten verdeutlicht. Aufgrund technologischer Abhängigkeiten kann eine tragende Stütze nur gebaut werden, wenn die darunterliegende, von ihr berührte Bodenplatte bereits fertiggestellt wurde. Ist hingegen der Realisierungszeitraum einer Platte dem der darauf ruhenden Stütze **nachgelagert**, stellt dies einen Widerspruch in der raum-zeitlich-semantischen Konsistenz des Modells dar und ein Fehler in

Modellierungsbereich	semantische Relation	Räumliche Analyse
Befestigungen von Elementen	IfcRelConnectsWithRealizingElements	Berühren die Befestigungen das Bauteil?
Kollisionen zwischen Elementen	IfcRelInterferesElements	Überschneiden sich die Bauteile? Berührt die InterferenceGeometry beide Bauteile?
Verbindungen zwischen Elementen	IfcRelConnectsElements	Berüht die ConnectionGeometry beide Bauteile?
Raumgrenzen durch Elemente	IfcRelSpaceBoundary	Berühren die Bauteile den Raum? Berühren die ConnectionGeometrie-Instanzen den Raum?

Tabelle 8.1: Weitere Möglichkeiten zur räumlich-semantischen Konsistenzprüfung von IFC-Modellen mit QL4BIM

der baulichen Terminplanung ist naheliegend. Auflistung 8.7 beinhaltet eine QL4BIM-Anfrage zur Selektion derartiger IfcSlab/IfcColumn-Paare.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 slab = TypeFilter (entities is IfcSlab)
3 columns = TypeFilter (entities is IfcColumn)
4 columns = PropertyFilter(columns.LoadBearing = True)
5 a[slab|column] = Touches(slabs, columns, 0.03)
6 b[slab|column] = AboveStrict(a)
7 c[slab|column] = After(b)
```

Auflistung 8.7: Raum-zeitlich-semantische Konsistenzprüfung mit QL4BIM, Anfrage 1 (Bodenplatten und Stützen)

Die Anfrage beginnt mit der Selektion aller Stützen und Bodenplatten (Z. 2 - 3). In Zeile 4 wird die Menge der Stützen auf tragende Instanzen eingeschränkt. Durch den `Touches`-Operator werden Paare aus sich berührender Platte und Stütze gebildet, wobei eine Toleranz von $\frac{3}{100}$ Einheiten angewendet wird (Z. 5). Um Plattenberührungen durch Stützen von unten auszuschließen, werden in der nächsten Anweisung die Paare über den `AboveStrict`-Operator reduziert. Die zeitliche Überprüfung erfolgt im letzten Schritt der Anfrage in Zeile 7. Der `After`-Operator identifiziert Paare, in denen die Bodenplatte **nach** der Stütze erstellt wird. Damit genügen diese Paare nicht der untersuchten raum-zeitlich-semantischen Konsistenz.

In der nächsten Konsistenzprüfung wird davon ausgegangen, dass nicht-tragende Wände innerhalb eines Raums erst erstellt werden können, wenn der Bodenbelag, beispielsweise Estrich, bereits fertiggestellt wurde. Beläge können im IFC-Datenmodell durch IfcCovering vorgehalten werden. Auflistung 8.8 beinhaltet eine QL4BIM-Anfrage zur Selektion derartiger IfcCovering/IfcWall-Paare.

```
1  entities = ImportModel("C:\TranslaTUM.ifc")
2  spaces = TypeFilter (entities is IfcSpace)
3  coverings = TypeFilter (entities is IfcCovering)
4  coverings = AttributeFilter(coverings.PredefinedType = Flooring)
5  a[space|covering]= Contains(spaces, coverings)
6  b[space|covering|element] = Deassociater(a, [space].ContainsElements)
7  c[covering|element] = Projector(b, [covering], [element])
8  d[covering|wall] = TypeFilter(c, [element] is IfcWall)
9  e[covering|wall] = PropertyFilter(d, [wall].LoadBearing = False)
10 f[covering|wall] = After(e)
```

Auflistung 8.8: Raum-zeitlich-semantische Konsistenzprüfung mit QL4BIM, Anfrage 2 (nicht-tragende Wände und Bodenbeläge)

Zu Beginn der Anfrage werden Entitäten der Typen IfcSpace und IfcCovering selektiert (Z. 2-3). Durch einen Attributfilter wird die mengenbasierte Variable `coverings` auf Bodenbeläge eingeschränkt. Über den `Contains`- und den `Deassociater`-Operator wird anschließend die Relation `b[space|covering|element]` erstellt (Z. 5 - 6). Deren Tripel setzen sich aus Raum, enthaltenem Belag und enthaltenem Bauelement zusammen. Da der Raum nicht weiter ausgewertet werden muss, kann dieser durch eine Projektion entfernt werden. In den Zeilen 8 und 9 werden die Bauelemente auf den Typ IfcWall und auf das Attribut *LoadBearing* hin gefiltert. Im letzten Schritt der Anfrage identifiziert der `After`-Operator die Paare, in denen der Bodenbelag **nach** der nicht-tragenden Wand erstellt werden soll (Z. 10). Diese Paare genügen somit nicht der untersuchten raum-zeitlich-semantischen Konsistenz.

Die vorhergehenden Anfragen zeigten auf, dass sich durch QL4BIM-Operatoren ansonsten getrennte Modellierungsbereiche untereinander abgleichen lassen. Dadurch wird die räumlich-semantische und die raum-zeitlich-semantische Konsistenz von IFC-Modellen in vielfacher Hinsicht überprüfbar. Zudem verdeutlichen die diskutierten Konsistenzprüfungen, dass in nicht-trivialen Anfragen die Erstellung und die Weiterverarbeitung relationaler Daten erforderlich sind. Durch das entwickelte Typsystem der Anfragesprache ist die Unterstützung eines entsprechenden Datentyps voll gegeben.

8.1.3 Teilmodellextraktion und Modellintegration

In Kapitel 5.3 wurde auf die Kollaboration zwischen Fachplanern eingegangen. Hier ergeben sich mitunter aufwendige Arbeitsschritte bezüglich der Modellextraktion und Modellreintegration, die in derzeit verwendeten BIM-Anwendungen manuell ausgeführt werden müssen. Im Folgenden wird beschrieben, wie die Extraktion von Teilmodellen und die Reintegration

von Modellen in die Gesamtdatenbasis durch den Einsatz von QL4BIM automatisiert werden können.

Auflistung 8.9 demonstriert, wie mit dem in Kapitel 5.3.1 entwickelten `MvdFilter` eine Teilmodellextraktion in QL4BIM erfolgen kann. Dazu wird im ersten Schritt die Gesamtdatenbasis durch den Import einer IFC-Datei geladen. Dem nachfolgenden `MvdFilter` werden alle Entitäten und der Pfad zu einer mvdXML-Datei übergeben (Z. 2). Auf Basis des MVDs erfolgt nun einer Selektion aus der `entities`-Menge. Durch den Aufruf des Operators wird zusätzlich eine Report-Datei erstellt, die Auskunft gibt, ob alle im MVD genannten Informationen verfügbar waren. In Zeile 3 werden die selektierten Entitäten in eine eigenständige IFC-Datei exportiert.

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 entities = MvdFilter(entities,"C:\IFC4ReferenceView.mvdxml")
3 entities = ExportModel(entities,"C:\TranslaTUM_RV.ifc")
```

Auflistung 8.9: QL4BIM-Anfrage zur Extraktion eines Teilmodells durch MVD-Verarbeitung

Auflistung 8.10 zeigt eine QL4BIM-Anfrage zur Teilmodellextraktion ohne die Verwendung eines MVDs. Hier erfolgt die Filterung durch wiederholte Typfilterung und Vereinigung der so erhaltenen Entitätsmengen. Nach dem Import einer IFC-Datei werden in den Zeilen 2 bis 5 jeweils Stützen-, Wände-, Platten- und Dach-Instanzen selektiert. In Zeile 6 werden die einzelnen Mengen mit Hilfe des `Union`-Operators zu einer Gesamtmenge vereinigt. Zuletzt werden die selektierten Entitäten als eine eigenständige IFC-Datei exportiert (Z. 7).

```
1 entities = ImportModel("C:\TranslaTUM.ifc")
2 columns = TypeFilter(entities is IfcColumn)
3 walls = TypeFilter(entities is IfcWall)
4 slabs = TypeFilter(entities is fcSlab)
5 roofs = TypeFilter(entities is IfcRoof)
6 someEntities = Union(columns, walls, slabs, roofs)
7 exportedEntities = ExportModel(someEntities, "C:\TranslaTUM_Sub1.ifc")
```

Auflistung 8.10: QL4BIM-Anfrage zur Extraktion eines Teilmodells durch wiederholte Typfilterung

Die so gewonnenen Teilmodelle müssen nach deren Bearbeitung und Erweiterung in den Gesamtdatenbestand des Projekts integriert werden. In Kapitel 5.3.1 wurde hierfür ein Operator nach dem Prinzip des three-way mergings entwickelt. Auflistung 8.11 demonstriert wie mit dem `Merger`-Operator eine Modellreintegration in QL4BIM erfolgen kann.

```
1 entitiesD0 = ImportModel("C:\TranslaTUM_V10.ifc")
2 entitiesD1 = ImportModel("C:\TranslaTUM_V11.ifc")
3 entitiesD2 = ImportModel("C:\TranslaTUM_MEP.ifc")
4 entities = Merger(entitiesD0, entitiesD1, entitiesD2)
5 exportedEntities = ExportModel(entities, "C:\TranslaTUM_Merged.ifc")
```

Auflistung 8.11: QL4BIM-Anfrage zum three-way merging zweier bearbeiteter Modelle

Zu Beginn der Anfrage werden drei IFC-Dateien geladen, die die Datensätzen D_0, D_1 und D_2 repräsentieren. Dem `Merger`-Operator in Zeile 4 werden die drei mengenbasierten Variablen übergeben. Der Operator führt einen three-way merge aus und liefert das Ergebnis als Menge an Entitäten zurück. Durch den Aufruf des Operators wird zusätzlich eine Report-Datei erstellt. Diese gibt Auskunft, ob die Integration fehlerfrei durchgeführt werden konnte oder ob Konflikte aufgetreten sind. In Zeile 5 wird die erhaltene Gesamtdatenbasis als eigenständige IFC-Datei exportiert.

Die entwickelten Operatoren zur Teilmodellextraktion unterstützen die Nutzung einer benutzerdefinierten Model View Definition. Außerdem können die weiteren Operatoren der Anfragesprache kombiniert werden, um ein Teilmodell zu gewinnen. Zur Reintegration von Modellen stellt die Anfragesprache einen three-way-merge-Operator bereit. Dieser macht sich die räumlichen Funktionen der Anfragesprache zu Nutzen und bietet dadurch eine erweiterte, von Entitätsbezeichnern unabhängige Modellreintegration. Derartige Funktionalität hebt sich deutlich von derzeit verfügbaren Methoden und der üblichen manuellen Arbeitsweise in diesem Bereich ab.

8.1.4 Anwendungsbeispiel zur kombinierten IFC-CityGML-Analyse

Wie in Kapitel 6.2.8 erörtert, ist QL4BIM für die Auswertung von IFC- und CityGML-Daten ausgelegt. Die Vorteile einer kombinierten Analyse werden im Folgenden anhand der Planung einer U-Bahn-Strecke diskutiert.

Für die kleinmaßstäbliche Planung der Trasse kommen CityGML-Daten zum Einsatz. Die Detailplanung neuer Bahnhöfe, Kreuzungspunkte und Sicherheitseinrichtungen erfolgt hingegen in bauteilbezogener IFC-Modellierung (Borrmann et al., 2015). Abbildung 8.4 zeigt beispielhafte Daten im Bereich der Münchner Innenstadt, wobei CityGML-Entitäten in grau und IFC-Entitäten in blau dargestellt werden.

Von Interesse sind beispielsweise Interaktionen zwischen den geplanten Untergrundbauwerken der U-Bahntrasse mit bereits existierenden Bauwerken. Speziell sollen aus der heterogenen Datenbasis zu jedem geplanten Raum darüber liegende Bestandsgebäude selektiert werden. Auflistung 8.12 zeigt die dafür nötige QL4BIM-Anfrage.

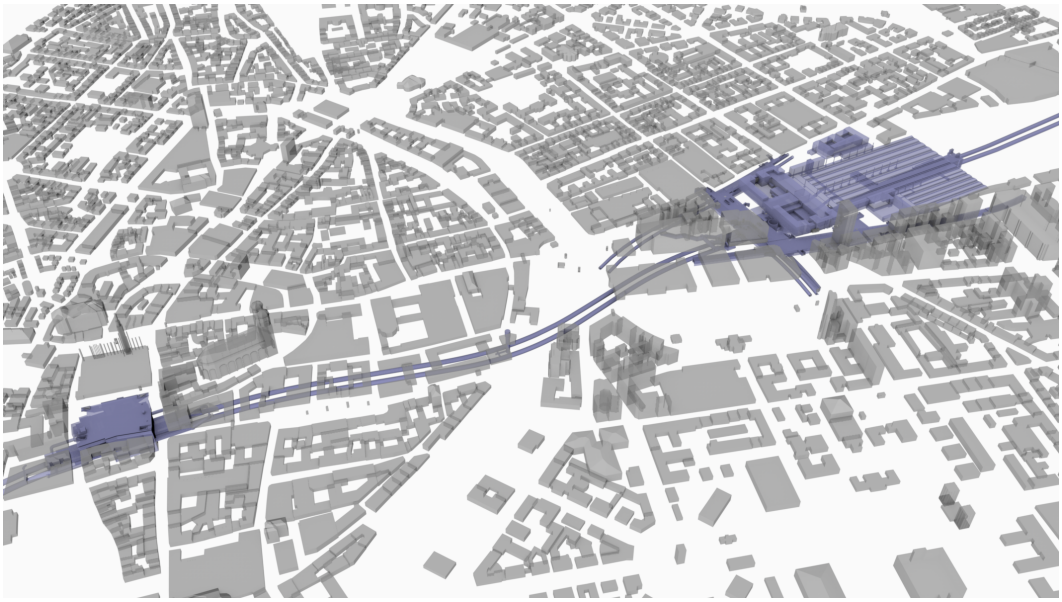


Abbildung 8.4: CityGML-Daten der Münchner Innenstadt in LoD1 in Kombination mit IFC-Daten einer geplanten U-Bahn-Trasse, Daten aus (Borrmann et al., 2015)

```
1 gmlEntities = ImportModel ("C:\Building_LOD1.gml")
2 gmlBuildings = TypeFilter (gmlEntities is Building)
3 ifcEntities = ImportModel ("C:\supportingBuildings.ifc")
4 ifcSpaces = TypeFilter (ifcEntities is IfcSpaces)
5 a[newSpace|oldBuilding] = AboveRelaxed (ifcSpaces, gmlbuildings)
```

Auflistung 8.12: Analyse direktionaler Relationen zwischen geplanten Räumen und Bestandsbauwerken in QL4BIM

Zu Beginn der Anfrage wird ein CityGML-Modell in den Data Pool des Laufzeitsystems importiert (Z. 1). Durch Anwendung des `TypeFilter`s in Zeile 2 werden alle CityGML-Entitäten vom Typ Building ausgewählt. Daraufhin werden die IFC-Daten der Bauplanung importiert und daraus IfcSpace-Instanzen selektiert (Z. 3 u. 4). Zur direktionalen Analyse werden dem `AboveRelaxed`-Operator die geplanten Räume und die Bestandsgebäude als Argumente übergeben (Z. 5). Die identifizierten Raum-Gebäude-Paare werden der relationalen Variable `a[newSpace|oldBuilding]` zugewiesen.

Die Anfrage in Auflistung 8.13 zeigt eine detailliertere Analyse der heterogenen Datenbasis aus IFC- und CityGML-Entitäten. Zu jeder geplanten Rohrleitung sollen Wände von Bestandsgebäuden selektiert werden. Dabei sind ausschließlich Gebäude mit mehr als zwei oberirdischen Etagen zu berücksichtigen. Zu selektieren sind diejenigen Wände dieser Gebäude, die sich innerhalb von drei Metern zur Leitung befinden.

In der Anfrage werden aus der geladenen CityGML-Datenbasis durch einen `TypeFilter` und einen `AttributeFilter` die entsprechenden Gebäude ausgewählt (Z. 2 u. 3). Danach erfolgt

die Dereferenzierung der Begrenzungen dieser Gebäude. In Zeile 5 werden die Begrenzungen durch einen `TypeFilter` auf WallSurfaces eingeschränkt. Danach erfolgt eine Projektion, da die Gebäudeinstanzen nicht mehr benötigt werden. In Zeile 7 und 8 werden eine IFC-Datei geladen und IfcFlowSegments selektiert. Durch den metrischen `Nearer`-Operator in Zeile 9 werden Paare aus Rohrleitung und Wandfläche gebildet, wenn die Distanz zwischen beiden Entitäten unter drei Metern beträgt.

```
1 gmlEntities = ImportModel ("C:\Building_LOD4.gml")
2 gmlBuildings = TypeFilter (gmlEntities is Building)
3 gmlBuildings = AttributeFilter (gmlBuildings.storeysAboveGround > 2)
4 a[gmlBuilding|gmlBound] = Dereferencer (highBuildings.boundedBy)
5 b[gmlBuilding|gmlWall] = TypeFilter (a, [gmlBound]  is WallSurface)
6 gmlWall = Projector (b, [gmlWall])
7 ifcEntities = ImportModel ("C:\supportinBuildings.ifc")
8 ifcFlowSegments = TypeFilter (ifcEntities is IfcFlowSegment)
9 c[ifcFlowSegment|gmlWall] = Nearer (ifcFlowSegments, gmlWall, 3)
```

Auflistung 8.13: Analyse metrischer Relationen zwischen IfcFlowSegment- und WallSurface-Entitäten in QL4BIM

Um Daten der Geoinformatik und des Bauingenieurwesens zusammen auswerten zu können, werden derzeit Konvertierungen zwischen den entsprechenden Formaten eingesetzt. Wegen der unterschiedlichen Ausrichtung beider Datenmodelle ist hierdurch jedoch der Verlust an Informationen wahrscheinlich, da nicht alle Gegebenheiten im jeweils anderen Schema abgebildet werden können. Um dies zu umgehen, unterstützt QL4BIM die simultane Verarbeitung von IFC- und CityGML-Entitäten. Als Basis fungiert hierfür das abstrahierende, interne Datenschema von QL4BIM. Dabei bleiben die schemaspezifischen Modellierungen erhalten und es wird keine Konvertierung zwischen den beiden Datenformaten durchgeführt. Gleichzeitig können Daten beider Schemata in einer integrierten Analyse verarbeitet und ausgewertet werden.

8.2 Untersuchung der Systemleistung

Als interaktive Anwendung zur Analyse umfangreicher Gebäudemodelle müssen Anfragen in QL4BIM in einem annehmbaren Zeitrahmen ausgeführt werden. In dieser Arbeit wird für Datenbasen in der Größenortung des TranslaTUM-Modells eine Ausführungszeit von 10 Sekunden als Höchstwert für nicht-räumliche Anfragen festgelegt. Bei der Auswertung von räumlichen Aspekten in derartigen Modellen wird eine Ausführungszeit von unter 20 Sekunden pro Statement angestrebt. Zur Überprüfung dieses Ziels wurde die QL4BIM-Sprache inklusive Laufzeitsystem nach den Vorgaben aus Kapitel 6 und Kapitel 7 als prototypische

BIM-Anwendung umgesetzt. Als Programmiersprache wurde C# verwendet und ein single-threading Ansatz verfolgt.

In empirischen Testläufen wurden die durchschnittlichen Ablaufzeiten für die vier QL4BIM-Musteranfragen ermittelt. Die gewonnenen Daten sind in Tabelle 8.2 dargestellt. Hierbei wurden die Zeiten für den Import der Entitäten inklusive Geometrie- und R-Tree-Erstellung nicht berücksichtigt. Eine ausführliche Darstellung der ermittelten Ausführungszeiten für jeden Einzelschritt der Anfragen ist im Anhang in Abschnitt A.12 enthalten.

Operation	# Input	# Output	Laufzeit (ms)
Musteranfrage 1	421040	2287	244
Musteranfrage 2	421040	442	232
Musteranfrage 3	421040	2	237
Musteranfrage 4	421040	1077	2866

Tabelle 8.2: Empirische Messungen der Ausführungszeit für Musteranfrage 1 - 4

In einer zweiten Auswertung wird eine Auswahl der räumlichen Operatoren betrachtet. Dafür wird die Anfrage in Auflistung 8.14 ausgeführt. Diese liefert Paare aus sich berührender Wand und Platte, Paare aus Stockwerk und beherbergter Platte sowie Paare aus Stockwerk und beherbergter Wand. Außerdem werden Paare aus Stockwerk und sich darüber befindender Wand gewonnen und Paare aus Wand und westlich liegender Stütze gebildet.

```
1  entities = ImportModel("C:\temp\translatumSt.ifc")
2  slabs = TypeFilter(entities is IfcSlab)
3  walls = TypeFilter(entities is IfcWall)
4  storeys = TypeFilter(entities is IfcBuildingStorey)
5  columns = TypeFilter(entities is IfcColumn)
6  a[wall|slab] = Touches(walls, slabs)
7  b[storey|slab] = Contains(storeys, slabs)
8  c[storey|wall] = Contains(storeys, walls)
9  d[storey|wall] = AboveRelaxed(storeys, walls)
10 e[wall|column] = WestRelaxed(walls, columns)
```

Auflistung 8.14: Anfrage mit den räumlichen Operatoren Contains, Touches, AboveRelaxed und WestRelaxed

Auf Basis der im Kapitel 7 vorgestellten Implementierungsstrategie wurde eine prototypische Implementierung der Query Language for 4D Building Information Models erstellt. Die damit durchgeführten Laufzeitmessungen zeigen, dass ein BIM-Analysesystem erstellt werden konnte, das praktischen Anforderungen hinsichtlich der Ausführungsgeschwindigkeit genügt. Die räumlichen Auswertungen sind, wie zu erwarten, die rechenintensivsten Analysen. Diese profitieren von der direkten BRep-

Operation	# Input (spatial)	# Output	Laufzeit (sec)
Touches	123012	1339	03.493
Contains (1)	486	28	00.184
Contains (2)	20583	766	17.416
AboveRelaxed	20583	5804	18.227
WestRelaxed	663230	7325	02.061

Tabelle 8.3: Empirische Messungen der Ausführungszeit zur Anfrage in Auflistung 8.14

Verarbeitung und der räumlichen Indexierung mit R*-Trees. Dies zeigt sich in der Ausführungsgeschwindigkeit und dem Skalierungsverhalten bei Anfragen mit räumlichen Operatoren, siehe dazu auch (Daum & Borrmann, 2014). Die absoluten Ausführungszeiten lassen sich zudem durch eine Implementierung in einer nativen Programmiersprache wie C, C++ oder Fortran und Multithreading zusätzlich reduzieren.

8.3 Vergleich des grammatikalischen Umfangs

Alle in dieser Arbeit behandelten allgemeinen Anfragesprachen basieren auf Sprachgrammatiken die in der Extended Backus-Naur Form verfügbar sind. Bei der Entwicklung von QL4BIM wurde versucht, den Umfang der Grammatik möglichst gering zu halten. Es wird davon ausgegangen, dass sich durch die wenigen Sprachkonstrukte das Erlernen der Sprache für Fachanwender einfacher gestaltet.

EBNF-Grammatiken sind im Allgemeinen nicht unbedingt auf einen geringen Umfang ausgelegt. Häufig werden aus technischen Gründen mehr Produktionen definiert als grammatikalisch nötig. Dennoch liefert die Produktionsanzahl einer Sprache ein ungefähres Maß für den Umfang an Sprachkonstrukten. Daher wird in Tabelle die Anzahl an Produktionen von zwei SQL-Varianten, XQuery und SPARQL mit der von QL4BIM verglichen.

Sprache	Produktionen	Referenz
SQL 2003	1312	(Leffler & Savage, 2017)
SQL 1999	1261	(Leffler & Savage, 2017)
XQuery	196	(W3C, 2014d)
SPARQL 2003	100	(W3C, 2008)
QL4BIM	25	Auflistung 6.16

Tabelle 8.4: Anzahl der Produktionen in allgemeinen Anfragesprachen und QL4BIM

QL4BIM kommt im Vergleich zu allgemeinen Anfragesprachen mit einem Bruchteil an Produktionen aus. Dies wird als Indiz gewertet, dass sich QL4BIM für Anwender ohne Programmierkenntnisse leichter erlernen lässt als die genannten allgemeinen Anfragesprachen.

8.4 Benutzerbefragung

Im letzten Teil der Validierung wird untersucht, ob QL4BIM aus Sicht der Endnutzer eine erleichterte Anwendbarkeit gegenüber den verfügbaren Ansätzen bietet. Dazu wurde unter 25 Master-Studentinnen und -Studenten der Studiengänge Bauingenieurwesen, Umweltingenieurwesen und Computational Mechanics im Kurs *Building Information Modeling* eine Umfrage durchgeführt. Dabei bewerteten die Studenten die Verständlichkeit der folgenden zwei Anfragen:

- Es sollen alle Türen selektiert werden, die höher als 200 Zentimeter sind und den OperationType REVOLVING aufweisen.
- Es soll jede Wand und das Stockwerk, in dem sie sich befindet, selektiert werden. Dabei muss das Stockwerk mindestens auf einer Höhe von 4 Metern liegen.

In der Umfrage wurden beide Anfragen als SQL- und ^{t}QL4BIM-Anfragen präsentiert und verglichen. Außerdem wurde die erste Anfrage in ^{v}QL4BIM umgesetzt und der SQL-Variante gegenübergestellt. Aus den abgegebenen Bewertungen für SQL und QL4BIM ergeben sich die in Abbildung 8.5 dargestellten Präferenzen. Die komplette Umfrage, in der Form wie sie den Studenten ausgegeben wurde, ist im Anhang in Abschnitt A.13 enthalten.

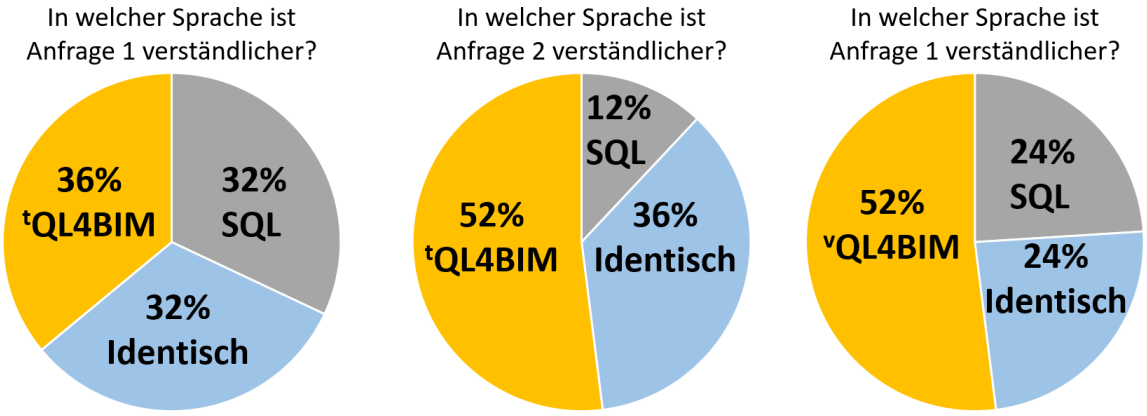


Abbildung 8.5: Benutzerumfrage zu Präferenz zwischen SQL und ^{t}QL4BIM bzw. ^{v}QL4BIM

Die durchgeführte Benutzerbefragung kann nicht als repräsentative Untersuchung gewertet werden. Dafür ist die Anzahl der Probanden zu gering und eine ausführlichere Untersuchung nötig. Dennoch scheint sich eine Tendenz zu QL4BIM sowohl in der textuellen als auch in der visuellen Variante gegenüber SQL abzuzeichnen. Dies deckt sich mit dem reduzierten grammatikalischen Konzept, das für QL4BIM entwickelt wurden.

8.5 Zusammenfassung

Dieses Kapitel untersucht die Eignung von QL4BIM und dem Laufzeitsystem der Sprache in den Bereichen Funktionsumfang, Leistung des Laufzeitsystems, grammatikalischer Umfang und Benutzerakzeptanz.

Als erstes wurde der erweiterte Funktionsumfang der Anfragesprache gegenüber verfügbaren Ansätzen aufgezeigt und dabei die 4D-Analyse, die Konsistenzprüfung von Modellen und die Modellextraktion/Modellreintegration betrachtet. Durch empirische Messungen konnte außerdem dargestellt werden, dass das Laufzeitsystem in der Lage ist, umfangreiche Modelle in der Größenordnung des TranslaTUM Projekts in annehmbarer Zeit zu analysieren. Der Vergleich des grammatikalischen Umfangs von zwei SQL-Varianten, XQuery und SPARQL mit dem von QL4BIM ergab, dass die vorgestellte Sprache mit einem Bruchteil an Produktionen auskommt. Dies wird als Indiz zur einfachen Anwendung der Sprache gewertet. Eine Benutzerumfrage zeigte zudem, dass Fachanwender der Sprache positiv gegenüberstehen.

Insgesamt kann somit davon ausgegangen werden, dass mit QL4BIM eine Anfragesprache entwickelt wurde, die die Analyse, Überprüfung und Weiterverarbeitung von 4D-Gebäudemodellen auf einem – sowohl konzeptionell als auch technisch – deutlich verbesserten Entwicklungsstand ermöglicht.

Kapitel 9

Zusammenfassung und Ausblick

In dieser Arbeit wurden Konzepte zur computergestützten Analyse und Verifikation von 4D-Gebäudemodellen entwickelt. Als Grundlage wurden die Prinzipien der Datenmodellierung im Building Information Modeling und im angrenzenden Fachgebiet Geoinformatik diskutiert. Anschließend kamen etablierte Anfragesprachen zur Verarbeitung von BIM-Daten zum Einsatz, wodurch sich Defizite dieser verfügbaren Methoden zeigten. Um diese zu überwinden, wurde die Anfragesprache QL4BIM und ein entsprechendes Laufzeitsystem konzipiert. Die Entwicklung berücksichtigte domänenspezifische Anforderungen mit dem Ziel, die Analyse und Verifikation von 4D-Gebäudemodellen auf einem deutlich höheren technologischen Niveau zu ermöglichen. Zur Validierung des vorgeschlagenen Ansatzes wurden die weitreichenden Einsatzmöglichkeiten der Sprache aufgezeigt und eine Benutzerbefragung durchgeführt. Eine prototypische Implementierung der Sprache und des Laufzeitsystems ermöglichten Aussagen zur Performance des Analysesystems.

9.1 Zusammenfassung

In den letzten Jahren haben sich mit der Etablierung von BIM die Modellierungstechniken im Bauwesen merklich weiterentwickelt. Aus Sicht der Bauinformatik fehlen jedoch teilweise Konzepte und entsprechende Werkzeuge innerhalb einer BIM-basierten Arbeitsumgebung. Dies hat nachteilige Auswirkungen auf die Wertschöpfung im Bauwesen und ist mit den steigenden Anforderungen an Bauwerke, deren effizienter Planung und Bauausführung nicht vereinbar.

Die Geoinformatik befasst sich mit Methoden zur Bereitstellung und Verarbeitung von Geodaten. Dies umfasst Ansätze für die räumliche sowie zeitliche Datenmodellierung und Verfahren zur räumlichen Indizierung. Geoinformationssysteme stellen etablierte Werkzeuge zur semantischen, geometrischen und topologischen Auswertung räumlicher Entitäten dar. Mit dem offenen Datenmodell CityGML für die Repräsentation von Stadtmodellen ist außerdem

die zunehmende Unterstützung dreidimensionaler Geometriedaten ersichtlich. Durch die detaillierte Betrachtung der genannten Themengebiete konnten Überschneidungen zur modellbasierten Planung im Bauwesen identifiziert und neuartige räumliche Analysefunktionen für BIM-Daten abgeleitet werden.

Eine herstellerneutrale Beschreibung von Gebäudemodellen ist über das offene IFC-Format möglich. Das EXPRESS- und XML-basierte, hoch-strukturierte Datenmodell kombiniert attributive und räumliche Daten der Entitäten eines Bauwerks. Zudem können unter diesen auftretende Beziehungen und planungsrelevante zeitliche Informationen der Terminplanung vorgehalten werden. Zur Realisierung des Datenmodells wird ein vierschichtiges Modellierungskonzept eingesetzt. Dieses besteht aus domänenspezifischen, domänenübergreifenden und grundlegenden Entitätsdefinitionen sowie einer untergeordneten Ressourcenmodellierung. Durch eine parallele Vorhaltung von semantischen, räumlichen und zeitlichen Daten, die Verwendung von expliziten Relationsobjekten, dynamisch zuweisbaren Merkmalslisten und vielfältigen Geometrierepräsentationen ermöglicht die IFC die Erstellung von Gebäudemodellen mit erweitertem Informationsgehalt. Im Gegensatz zur zwei- und dreidimensionalen CAD-Planung bietet sich damit ein hohes Potential an Qualitäts- und Effizienzsteigerung. Dieses kann jedoch nur ausgeschöpft werden, wenn Modelle durch die Fachanwender auf ihre Korrektheit analysiert, für unterschiedliche Arbeitsaufgaben vorbereitet und rekombiniert werden können.

Zur Analyse und Verarbeitung strukturierter Daten im Ingenieurwesen sind Datenbanken und ihre Anfragesprachen wichtige Werkzeuge. Im Gegensatz zum GIS-Bereich werden Anfragesprachen in BIM-Umgebungen jedoch selten eingesetzt. Was deren Einsatz behindert, wurde methodisch durch die Implementierung von Musteranfragen in unterschiedlichen Sprachen untersucht. Betrachtet wurden hierbei SQL, LINQ, XQuery, SPARQL und domänenspezifische Entwicklungen. Es konnte festgestellt werden, dass die Analyse und sprachbasierte Verarbeitung von IFC-Daten spezielle Anforderungen aufweist, die in keiner der behandelten Ansätze erfüllt sind. Im Speziellen ist die fehlende Berechnung räumlicher Prädikate erkennbar. Auch die Untersuchung von Vererbungsbeziehungen und inversen Attributen gestaltet sich umständlich bzw. unmöglich. Typkonvertierungen erhöhen zudem die Komplexität von Anfragen. Mit Ausnahme von XQuery muss zur Nutzung der genannten Anfragesprachen eine Transformation der IFC-Daten in das jeweilige Datenmodell erfolgen. Da diese Transformationen nicht standardisiert sind, lassen sich Anfragen in SQL, LINQ und SPARQL nur eingeschränkt über Systemgrenzen wiederverwenden. Dies stellt ein weiteres Hindernis für den Einsatz der bestehenden Ansätze dar.

Die Untersuchung der genannten etablierten Anfragesprachen ergab, dass sich aus Sicht des Endnutzers bereits inhaltlich überschaubare Analysen nur durch komplexe Anfragen realisieren lassen. In diesen treten syntaktische und konzeptionelle Feinheiten auf, die sich für den Fachanwender nicht ohne Weiteres erschließen. Auch die verfügbaren domänenenspezifischen

Konzepte ermöglichen keine BIM-Analyse auf hohem Abstraktionsniveau. Ausgehend von den erkannten Defiziten wurde die Anfragesprache QL4BIM für die Analyse, Verifikation und Weiterverarbeitung von BIM-Daten konzipiert. Im ersten Schritt wurden dazu zeitliche und räumliche Operatoren sowie Ansätze zur Teilmodellextraktion und Modellreintegration entwickelt. Im Falle der räumlichen Operatoren wurde insbesondere auf die benötigten algorithmischen Grundlagen zur effizienten Verarbeitung von dreidimensionalen Entitätsrepräsentationen eingegangen. Topologische und direktionale Operatoren konnten außerdem so erweitert werden, dass benutzerdefinierte Toleranzen unterstützt werden. Durch eine Reihe beispielhafter Anfragen konnte gezeigt werden, dass derartige Operatoren einen hohen technischen Nutzen bieten und in vielfältigen Fragestellungen eingesetzt werden können.

Die Query Language for 4D Building Information Models ist für den Einsatz durch Fachanwender ausgelegt, basiert auf einer überschaubaren Anzahl an grammatikalischen Regeln und erleichtert die Inspektion von Zwischenergebnissen. Die Sprache folgt dem imperativen Programmierparadigma, beinhaltet jedoch keine Kontrollstrukturen. Das flexible Typsystem der Sprache aggregiert Typen externer Schemata und lässt sich ohne Anpassung anderer Komponenten auf zusätzliche Datenformate erweitern. Domänenspezifische Operatoren verbergen die algorithmische Komplexität, die zur Analyse von BIM-Daten nötig ist. Somit können semantische, relationale, räumliche und zeitliche Aspekte der Datenbasis auf hohem Abstraktionsniveau ausgewertet werden. Für die formale Definition der Sprache in ihrer textuellen und visuellen Notation wurden zwei kontextfreie Grammatiken entwickelt. Zudem ist die Anfragesprache so konzipiert, dass Anfragen komplett aber auch schrittweise ausgeführt werden können. Im schrittweisen Modus werden Zwischenergebnisse präsentiert. Dies soll die Anfrageerstellung erleichtern und verdeutlicht die Ausrichtung der Sprache auf Ingenieure und Architekten.

Zur Ausführung von QL4BIM wird ein geeignetes Laufzeitsystem benötigt. Das im Rahmen dieser Arbeit konzipierte System interpretiert Anfragen und stellt die beschriebenen Sprachoperatoren bereit. Außerdem hält es die zu untersuchende Datenbasis in einer für die Analyse vorteilhaften Repräsentation vor. So werden die geometrischen Daten als BRep-Strukturen mit R*-Trees indiziert. Hierdurch können metrische, direktionale und topologische Untersuchungen effizient ausgeführt werden. Das Laufzeitsystem synchronisiert zudem die textuelle und visuelle Notation der Anfragesprache. Der Anwender kann dadurch jeder Zeit zwischen beiden Schnittstellen des Anfragesystems wechseln. Die Funktionalität wird durch die Verarbeitung eines abstrakten Syntaxbaums realisiert, der nicht nur zur Anfrageausführung sondern auch zur Anfragegenerierung in beiden Notationen dient.

Die entwickelte raum-zeitliche Anfragesprache für die Prüfung und Analyse von 4D-Bauwerksmodellen wurde hinsichtlich mehrerer Kriterien validiert. Zum einen wurde die Sprache und ihr Laufzeitsystem prototypisch implementiert. In der Applikation konnten alle Musteranfragen umgesetzt werden. Empirische Laufzeituntersuchung ergaben, dass die entwickelten Metho-

den den zuvor definierten Anforderungen an ein interaktives Anfragesystem gerecht werden. Eine Reihe weiterer beispielhafter Anfragen verdeutlicht den generellen Ansatz von QL4BIM. Behandelt wurden hier 4D-Analysen, die Prüfung der Modellkonsistenz, die Teilmodellerstellung und die Reintegration von Modellen. Zudem wurde die Möglichkeit einer kombinierten BIM-GIS-Analyse durch die Integration von Stadtmodellen aufgezeigt. Eine Umfrage unter Studenten zeigte die positive Einstellung von Fachanwendern zu ^{t}QL4BIM und ^{v}QL4BIM im Vergleich zur SQL. Dabei wurde nach einer kurzen Einführung erfragt, wie verständlich sich Analysen in beiden Sprachen bzw. Notationen darstellen. Die Ergebnisse der Umfrage decken sich mit dem Vergleich des grammatikalischen Umfangs von QL4BIM zu allgemeinen Anfragesprachen wie SQL.

Die im Zuge dieser Arbeit gewonnenen und hier nochmals zusammengefassten Erkenntnisse decken sich somit mit den in Kapitel 1.1 aufgestellten Thesen.

9.2 Ausblick

Zeitgleich mit der Einreichung dieser Arbeit wird QL4BIM als Open Source Software zur Verfügung gestellt. Das hohe Interesse an einer Nutzung der Anfragesprache in Forschungsvorhaben und kommerziellen Projekten zeigt den Bedarf einer entsprechenden Entwicklung und die Relevanz des Forschungsprojekts. Außerdem lässt es auf den regen Einsatz der Anfragesprache und des Laufzeitsystems hoffen. Dadurch werden sich neue Anforderungen ergeben, die zusätzliche Anstrengungen bedürfen.

Auch bevor QL4BIM in weiteren Projekten zum Einsatz kommt, lassen sich bereits zukünftige Forschungsbereiche ausmachen. So kann die Sprache zur Bearbeitungssprache erweitert werden, in der Entitäten der Datenbasis verändert und neu erstellt werden können. Dadurch kann die Automatisierung wiederkehrender Bearbeitungsschritte innerhalb einer BIM-Anwendung erreicht werden. Derartige Operatoren zur Bearbeitung und Erstellung von BIM-Entitäten lassen sich zudem vorteilhaft mit der bereits entwickelten Analysefunktionalität kombinieren.

Für eine Anfragesprache wie QL4BIM ist außerdem die Komplexität der verwendeten Datenschemata zu berücksichtigen. So muss der Endnutzer trotz der überschaubaren QL4BIM-Syntax Hintergrundwissen über die IFC und CityGML besitzen. Eine Abhilfe können Operatoren zu Schemauntersuchung darstellen. Diese helfen Endnutzern Anfragen zu erstellen, auch wenn sie keine umfassende Erfahrung mit den eingesetzten Datenschemata haben.

Im Bereich der geometrischen Verarbeitung können zusätzliche Operatoren neuartige Analysen ermöglichen. Ein Beispiel hierfür ist die Identifikation geometrisch ähnlicher Bauteile und die Analyse von Non-Uniform Rational B-Splines, die mit IFC4 eingeführt wurden. Auch die Verwendung von IFC-Snippets erfordert zusätzliche Forschung. Dies sind Teilmodelle, die keine komplette Modellstruktur beinhalten. Deren Einführung ist für die nächste IFC-Version

angedacht. QL4BIM kann für die Erstellung dieser Snippets und deren Zusammenführung zu validen Gesamtmodellen erweitert werden.

Anhang A

A.1 Definition des Building Modules (CityGML)

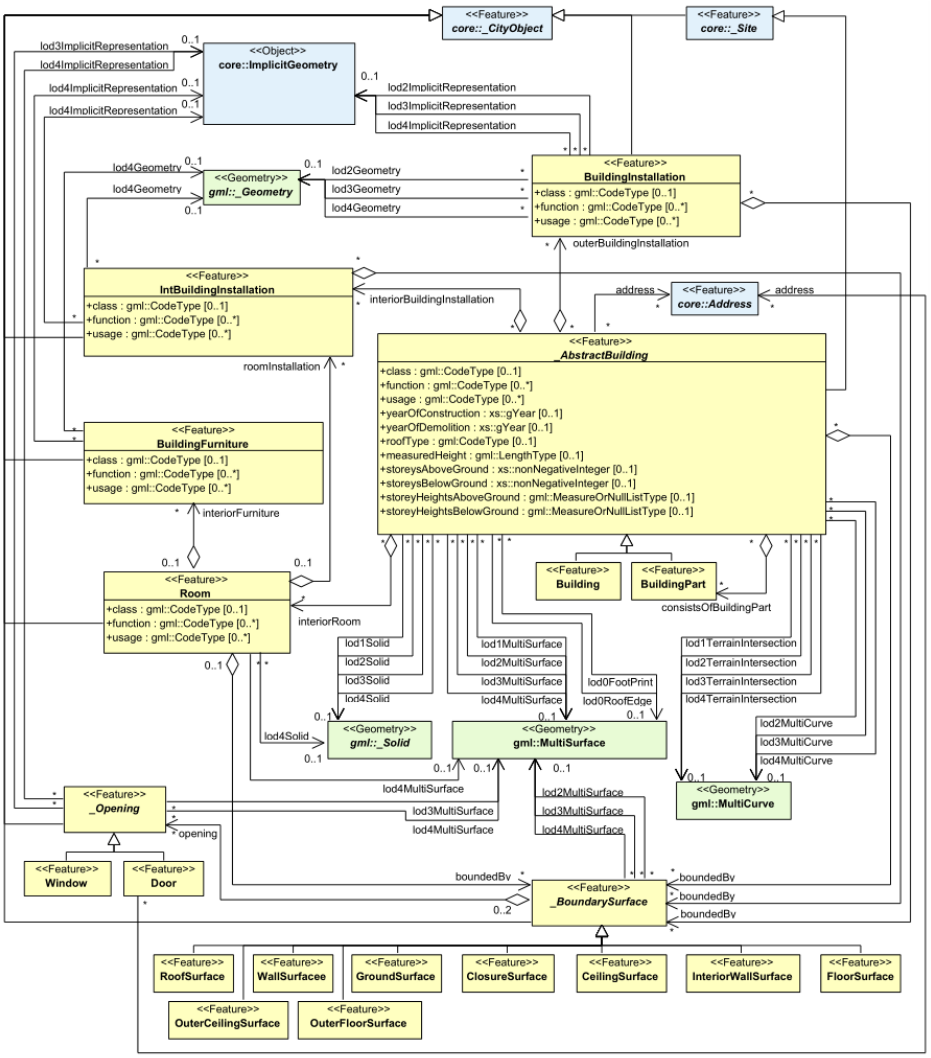


Abbildung A.1: UML-Klassendiagramm des Building Modules, aus (OGC, 2012, S. 63)

A.2 Topologische Definitionen

Im Folgenden sind die topologischen Definitionen von Egenhofer & Franzosa (1991, Kap. 3) aufgeführt.

- „Given $Y \subset X$, the *interior* of Y, denoted by Y°, is defined to be the union of all open sets that are contained in Y, i.e., the interior of Y is the largest open set contained in Y. y is in the interior of Y if and only if there is a neighborhood of y contained in Y, i.e. $y \in Y^\circ$, if and only if there is an open set U such that $y \in U \subset Y$. The interior of a set could be empty, e.g., the interior of the empty set is empty. The interior of X is itself. If U is open then $U^\circ = U$. If $Z \subset Y$ then $Z^\circ \subset Y^\circ$."

- „The *closure* of Y, denoted by $\overline{Y}$, is defined to be the intersection of all closed sets that contain Y, i.e., the closure of Y is the smallest closed set containing Y. It follows that y is in the closure of Y if and only if every neighborhood of y intersects Y, i.e., $y \in \overline{Y}$ if and only if $U \cap Y \neq \emptyset$ for every open set U containing y. The empty set is the only set with empty closure. The closure of X is X itself. If X is closed then $\overline{C} = C$. If $Z \subset Y$ then $\overline{Z} \subset \overline{Y}$."

- „The *boundary* of Y, denoted by ∂Y, is the intersection of the closure of Y and the closure of the complement of Y, i.e., $\partial Y = \overline{Y} \cap \overline{Y - Y}$. The boundary is a closed set. It follows that y is in the boundary of Y if and only if every neighborhood of y intersects both Y and its complement, i.e., $y \in \partial Y$ if and only if $U \cap Y \neq \emptyset$ and $U \cap (X - Y) \neq \emptyset$ for every open set U containing y. The boundary can be empty, e.g., the boundaries of both X and the empty set are empty."

A.3 Vererbungsgraph von IfcWallStandardCase

```
1  ENTITY IfcWallStandardCase
2  ENTITY IfcRoot
3    GlobalId        :IfcGloballyUniqueId;
4    OwnerHistory      :OPTIONAL IfcOwnerHistory;
5    Name          :OPTIONAL IfcLabel;
6    Description       :OPTIONAL IfcText;
7  ENTITY IfcObjectDefinition
8  INVERSE
9    HasAssignments      :SET OF IfcRelAssigns FOR RelatedObjects;
10   Nests         :SET [0:1] OF IfcRelNests FOR RelatedObjects;
11   IsNestedBy        :SET OF IfcRelNests FOR RelatingObject;
12   HasContext        :SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
13   IsDecomposedBy      :SET OF IfcRelAggregates FOR RelatingObject;
14   Decomposes        :SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
15   HasAssociations     :SET OF IfcRelAssociates FOR RelatedObjects;
16 ENTITY IfcObject
17   ObjectType        :OPTIONAL IfcLabel;
18 INVERSE
19   IsDeclaredBy      :SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
20   Declares        :SET OF IfcRelDefinesByObject FOR RelatingObject;
21   IsTypedBy       :SET [0:1] OF IfcRelDefinesByType FOR RelatedObjects;
22   IsDefinedBy       :SET OF IfcRelDefinesByProperties FOR RelatedObjects;
23 ENTITY IfcProduct
24   ObjectPlacement     :OPTIONAL IfcObjectPlacement;
25   Representation      :OPTIONAL IfcProductRepresentation;
26 INVERSE
27   ReferencedBy      :SET OF IfcRelAssignsToProduct FOR RelatingProduct;
28 ENTITY IfcElement
29   Tag           :OPTIONAL IfcIdentifier;
30 INVERSE
31   FillsVoids        : SET [0:1] OF IfcRelFillsElement FOR
     RelatedBuildingElement;
32   ConnectedTo       : SET OF IfcRelConnectsElements FOR RelatingElement;
33   IsInterferedByElements  : SET OF IfcRelInterferesElements FOR
     RelatedElement;
34   InterferesElements    : SET OF IfcRelInterferesElements FOR RelatingElement
     ;
35   HasProjections      : SET OF IfcRelProjectsElement FOR RelatingElement;
36   ReferencedInStructures  : SET OF IfcRelReferencedInSpatialStructure FOR
     RelatedElements;
37   HasOpenings       : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
38   IsConnectionRealization : SET OF IfcRelConnectsWithRealizingElements FOR
     RealizingElements;
39   ProvidesBoundaries    : SET OF IfcRelSpaceBoundary FOR
     RelatedBuildingElement;
```

```
40    ConnectedFrom      : SET OF IfcRelConnectsElements FOR RelatedElement;
41    ContainedInStructure  : SET [0:1] OF IfcRelContainedInSpatialStructure FOR
      RelatedElements;
42  ENTITY IfcBuildingElement
43  INVERSE
44    HasCoverings     :SET OF IfcRelCoversBldgElements FOR
      RelatingBuildingElement;
45  ENTITY IfcWall
46    PredefinedType     :OPTIONAL IfcWallTypeEnum;
47  ENTITY IfcWallStandardCase
48  END_ENTITY;
```

Auflistung A.1: Vererbungsgraph von IfcWallStandardCase

A.4 Die inversen Attribute der IfcElement-Entität

Inverses Attribut	Beschreibung
FillsVoids	Das Element füllt einen Leerraum auf.
ConnectedTo	Das Element ist mit einem weiteren Element logisch oder physikalisch verbunden.
IsInterferedByElements	Das Element überlagert sich mit einem weiteren Element.
InterferesElements	Wie IsInterferedByElements, das Element ist jedoch nicht Ausgangs- sondern Endpunkt der Verknüpfung.
HasProjections	Das Element wird über ein boolesche Operation (Union) mit einem weiteren Element erweitert.
ReferencedInStructures	Verknüpft das Element mit einer räumlichen Struktur. Dabei muss sich das Element nicht zwingend innerhalb der Struktur befinden. Diese Beziehung kann mehrfach pro Element genutzt werden.
HasOpenings	Erzeugt einen Leerraum im referenzierten Element.
IsConnectionRealization	Das Element erzeugt, ggf. mit anderen Elementen, eine Verbindung zwischen zwei weiteren Elementen.
ProvidesBoundaries	Definiert das Element als Begrenzung für einen Raum (IfcSpace).
ConnectedFrom	Wie ConnectedTo, das Element ist jedoch nicht Ausgangs- sondern Endpunkt der Verknüpfung.
ContainedInStructure	Verbindet das Element mit einer räumlichen Struktur. Die Beziehung kann nur einmal pro Element etabliert werden.

Tabelle A.1: Inverse Attribute der IfcElement-Entität

A.5 Prozessdiagramm

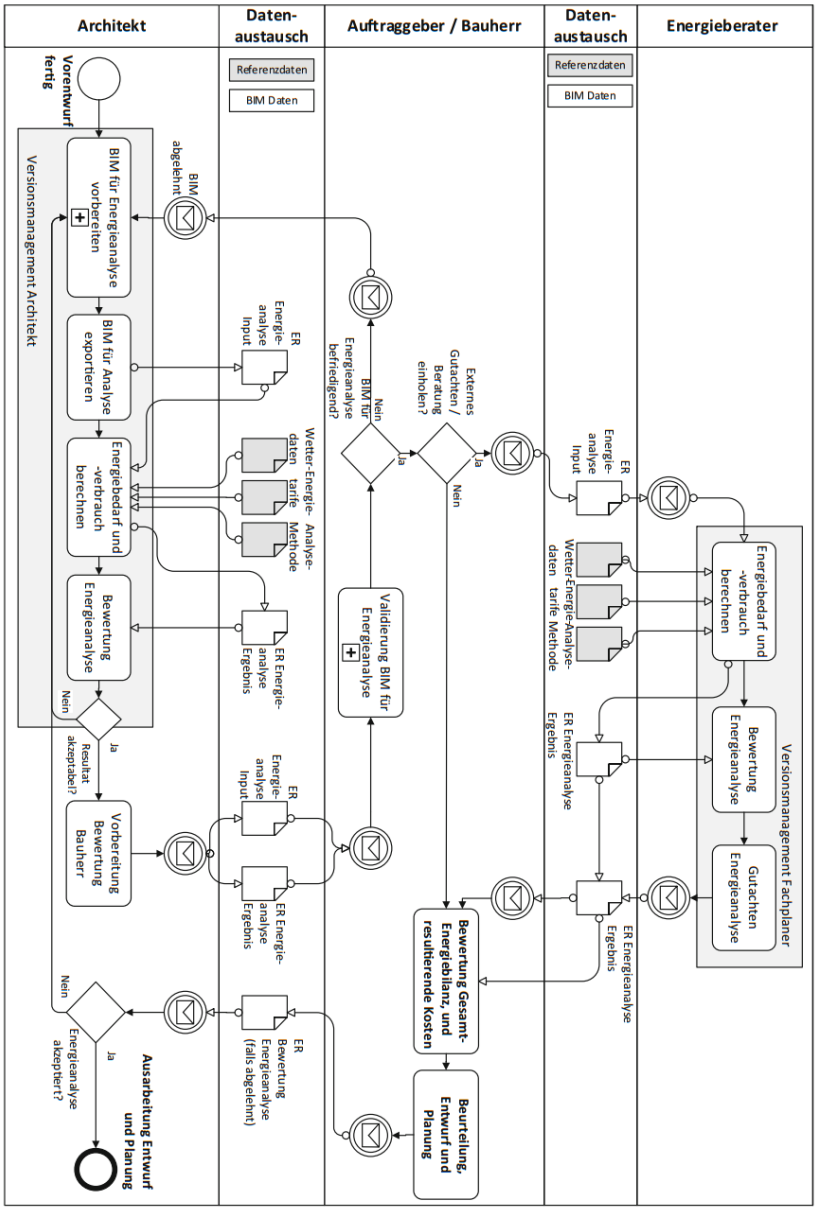


Abbildung A.2: Process Map Energy-Analysis aus (Borrmann et al., 2015, S. 135)

A.6 Relationsdefinitionen für ausgewählte IFC-Entitäten

IfcWallStandardCase(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar, Description:varchar, ObjectType:varchar, ObjectPlacement:FK, Representation:FK, Tag:varchar, PredefinedTyp:varchar)

IfcRelVoidsElement(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar, Description:varchar, RelatingBuildingElement:FK, RelatedOpeningElement:FK)

IfcOpeningElement(Id:int, GlobalId:varchar, OwnerHistory:FK, Name:varchar, Description:varchar, ObjectType:varchar, ObjectPlacement:FK, Representation:FK, PredefinedTyp:varchar)

IfcOpeningElement(Id:int, GlobalId:varchar, OwnerHistory, Name:varchar, Description:varchar, ObjectType, ObjectPlacement:FK, Representation:FK, PredefinedTyp)

IfcRelFillsElement(Id:int, GlobalId:varchar, OwnerHistory, Name:varchar, Description:varchar, RelatingOpeningElement:FK, RelatedBuildingElement:FK)

IfcDoor(Id:int, GlobalId:varchar, OwnerHistory, Name:varchar, Description:varchar, ObjectType, ObjectPlacement, Representation, Tag:varchar, OverallHeight:float, OverallWidth:float, PredefinedTyp:varchar, OperationType:varchar, UserDefinedOperationType:varchar)

Auflistung A.2: Relationsdefinitionen für IfcWallStandardCase, IfcRelVoidsElement, IfcOpeningElement, IfcRelFillsElement und IfcDoor entsprechend der zweiten Modellierungsvariante

A.7 Klassendefinitionen für LINQ-Anfragen in C#

```
1  public class IfcElement
2  {
3      public string Name { get; set; }
4
5      public IList<IfcRelVoidsElement> HasOpenings { get; set; }
6      public IfcRelFillsElement FillsVoids { get; set; }
7
8      public IfcElement()
9      {
10         HasOpenings = new List<IfcRelVoidsElement>();
11     }
12 }
13
14 public class IfcWallStandardCase : IfcElement {}
15
16 public class IfcDoor : IfcElement
17 {
18     public double OverallHeight { get; set; }
19     public Op OperationType { get; set; }
20 }
21
22 public class IfcOpeningElement : IfcElement
23 {
24     public IList<IfcRelFillsElement> HasFillings { get; set; }
25     public IfcRelVoidsElement VoidsElements { get; set; }
26
27     public IfcOpeningElement()
28     {
29         HasFillings = new List<IfcRelFillsElement>();
30     }
31 }
32
33 public class IfcRelVoidsElement
34 {
35     public IfcElement RelatingBuildingElement { get; set; }
36     public IfcOpeningElement RelatedOpeningElement { get; set; }
37
38     public IfcRelVoidsElement(IfcElement relatingBuildingElement,
       IfcOpeningElement relatedOpeningElement)
39     {
40         RelatingBuildingElement = relatingBuildingElement;
41         RelatedOpeningElement = relatedOpeningElement;
42         relatingBuildingElement.HasOpenings.Add(this);
43         relatedOpeningElement.VoidsElements = this;
44     }
```

```
45 }
46
47 public class IfcRelFillsElement
48 {
49     public IfcOpeningElement RelatingOpeningElement { get; set; }
50     public IfcElement RelatedBuildingElement { get; set; }
51
52     public IfcRelFillsElement(IfcOpeningElement relatingOpeningElement,
       IfcElement relatedBuildingElement)
53     {
54         RelatedBuildingElement = relatedBuildingElement;
55         RelatingOpeningElement = relatingOpeningElement;
56         relatingOpeningElement.HasFillings.Add(this);
57     }
58 }
```

Auflistung A.3: Klassendefinitionen der genutzten Entitäten in C#

A.8 Minimales XSD-Schema für Bauteile

```
1  <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
2  <xs:element name="building">
3   <xs:complexType>
4   <xs:sequence>
5    <xs:element name="wall">
6      <xs:complexType>
7      <xs:simpleContent>
8        <xs:extension base="xs:string">
9        <xs:attribute name="id" type="xs:int" />
10       </xs:extension>
11     </xs:simpleContent>
12     </xs:complexType>
13   </xs:element
14   <xs:element name="door">
15     <xs:complexType>
16     <xs:simpleContent>
17       <xs:extension base="xs:string">
18       <xs:attribute name="id" type="xs:int" />
19       </xs:extension>
20     </xs:simpleContent>
21     </xs:complexType>
22  </xs:sequence>
23  </xs:complexType>
24 </xs:element>
25 </xs:schema>
```

Auflistung A.4: Minimales XSD-Schema für die Aufzählung von Bauteilen eines Gebäudemodells

A.9 Die komplette Grammatik von vQL4BIM

```
[01] vQL4BIM ::= statement*
[02] statement ::= variable '=' expression
[03] expression ::= operator '(' argument ( ',' argument)* ')'
[04] variable ::= identifier | relation
[05] operator ::= [A-Z] (a-zA-Z0-9)*
[06] identifier ::= [a-z] (a-zA-Z0-9)*
[07] externalIdentifier ::= [^(".:<>=!~)]*
[08] argument ::= simpleType | identifier | relAttributeArgument |
                  attributeAccess | predicate
[09] relation ::= identifier identifier (identifier)+
[10] relAttributeArgument ::= identifier? '[' (number|identifier) ']'
[11] simpleType ::= number | float | string | logic
[12] sign ::= [+-]?
[13] number ::= sign [0-9]+
[14] float ::= sign [0-9]+ '.' [0-9]+ ('E' sign [0-9]+)?
[15] string ::= [^"]*
[16] logic ::= 'false' | 'true' | 'unknown'
[17] predicate ::= typePredicate | attributePredicate | countPredicate
[18] attributePredicate::= attributeAccess compare
                  (simpleType | attributeAccess)
[19] attributeAccess ::= setAttributeAccess | relAttributeAccess
[20] setAttributeAccess ::= identifier '.' externalIdentifier
[21] relAttributeAccess ::= relAttributeArgument '.' externalIdentifier
[22] typePredicate ::= (identifier | relAttributeArgument) 'is'
                  externalIdentifier
[23] countPredicate ::=  relAttributeArgument compare number
[24] compare ::= '<' | '>' | '>=' | '<=' | '!=' | '=' | '~'
```

Auflistung A.5: Die vQL4BIM-Definition in EBNF, ohne graphische Repräsentationen

A.10 Musteranfragen ohne Darstellung typisierter Konnektoren

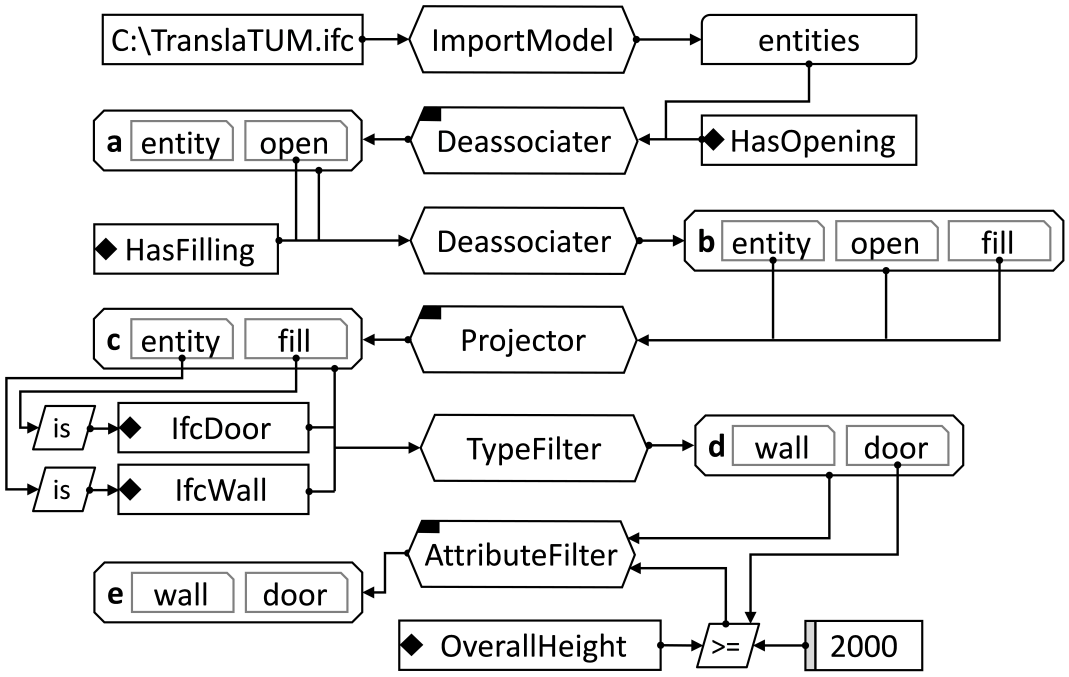


Abbildung A.3: Musteranfrage 2 (Wand/Tür Paare) in vQL4BIM, Konnektorendarstellung deaktiviert

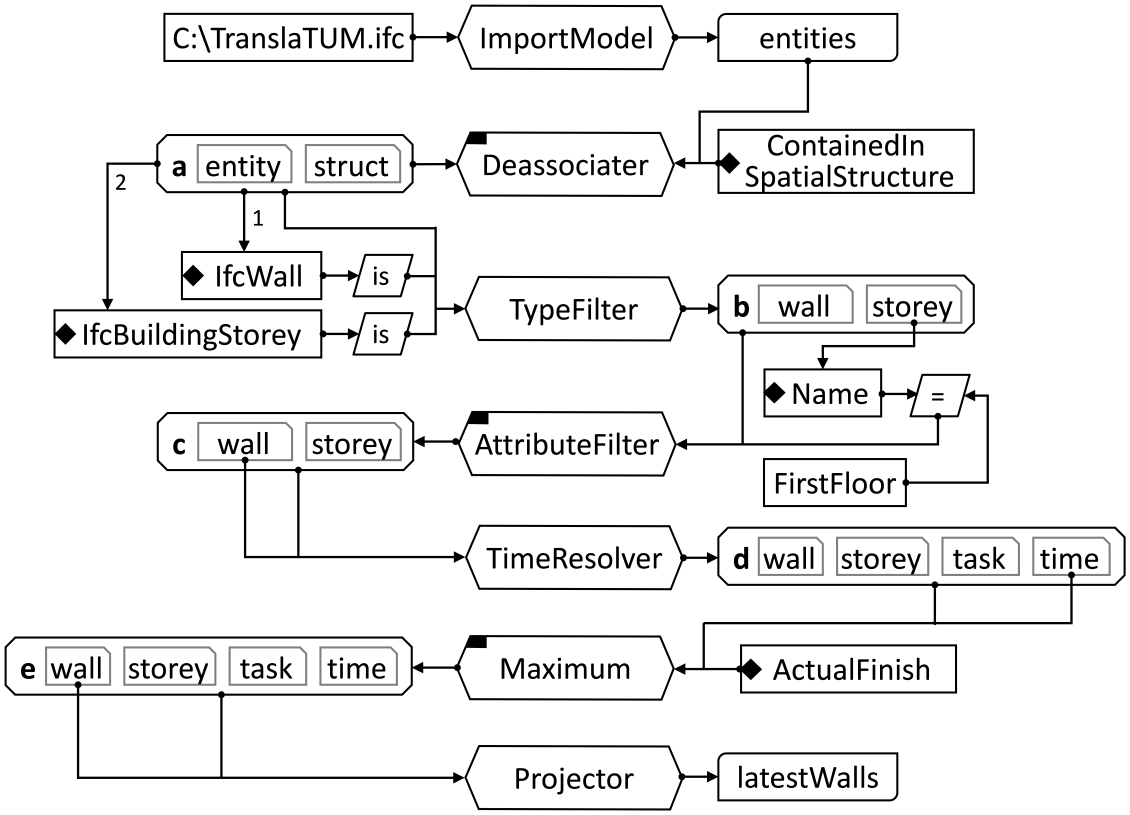


Abbildung A.4: Musteranfrage 3 (Zuletzt fertiggestellte Wände in Stockwerk) in ᵛQL4BIM, Konnektorendarstellung deaktiviert

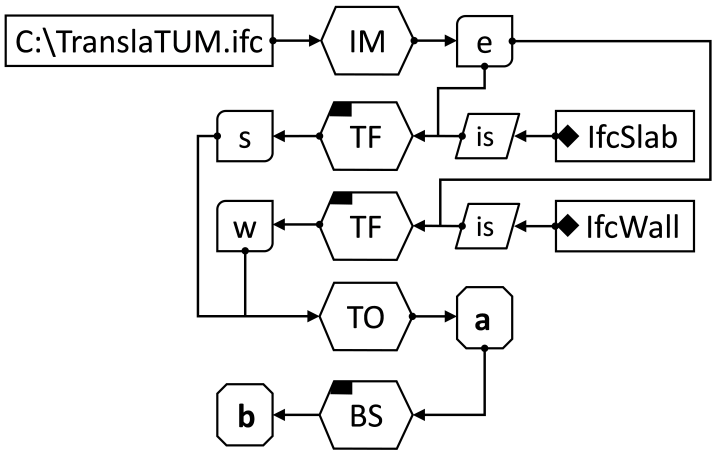


Abbildung A.5: Musteranfrage 4 (Paare aus Deckenplatte und berührender, darunterliegender Wand) in ᵛQL4BIM, kompakte Variante, Konnektorendarstellung deaktiviert

A.11 Prototypische Implementierung eines IFC Instance Parsers

```
1  using System.Text;
2  using QL4BIMinterpreter.QL4BIM;
3
4  COMPILER EXCHANGEFILE
5      public QLExchangeFile QLExchangeFile { get; } = new QLExchangeFile();
6
7  CHARACTERS
8    digit = "0123456789".
9    up = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
10   low = "abcdefghijklmnopqrstuvwxyz".
11   startLetter = up + low.
12   space = " ".
13   sharp = "#".
14   semicolon = ";".
15   text = up + low + digit + space.
16   stringDelimiter = semicolon + "\r" + "\n" + "'".
17   anyWithoutStringDelimiteres = ANY-stringDelimiter.
18
19 TOKENS
20   stringText = "'" {anyWithoutStringDelimiteres} "'".
21   entityId = sharp digit { digit }.
22   equal = "=".
23   obracket = '('.
24     cbracket = ')'.
25   float = ['+'|'-'] digit {digit} '.' {digit} ['E' ['+'|'-'] digit {digit}].
26   number = ['+'|'-'] digit {digit}.
27   indent = startLetter {text}.
28   star = "*".
29   semicolonT = semicolon {space} "\r\n".
30
31 PRODUCTIONS
32   /*=====================================================*/
33   classname <. out string name .>
34   = indent              (. name = t.val; .).
35
36   /*=====================================================*/
37   list<. out QLList QLList .>   (. QLList = new QLList(); .)
38   = '('                 (. QLPart QLPart; .)
39   part<out QLPart>          (. QLList.Add(QLPart); .)
40   {','
41   part<out QLPart>          (. QLList.Add(QLPart); .)
42   }
43   ')'.
44
```

```
45  /*=====================================================*/
46  myenum<out string enumstring> (. var sb = new StringBuilder(); .)
47  =
48  '.'                              (. sb.Append('.'); .)
49  {
50  ANY                (. sb.Append(t.val); .)
51  }
52  '.'                (. sb.Append('.');
53                      enumstring = sb.ToString(); .).
54
55  /*=====================================================*/
56  mystring<out string QLstring>   (. var sb = new StringBuilder(); .)
57  = stringText           (. sb.Append(t.val); .)
58                   (. QLstring = sb.ToString();  .).
59
60  /*=====================================================*/
61  part<out QLPart QLPart>   (. QLPart  = new QLPart();
62                     QLClass QLClass;
63                     QLList QLList;
64                     string QLstring;
65                     string enumstring; .)
66  = myenum<out enumstring>     (. QLPart.QLEnum = new QLEnum(enumstring); .)
67  | mystring<out QLstring>     (. QLPart.QLString = new QLString(QLstring); .)
68  | '$'                (. QLPart.IsNull = true; .)
69  | entityId             (. QLPart.QLEntityId = new QLEntityId(t.val); .)
70  | list<out QLList>          (. QLPart.QLList = QLList; .)
71  | float              (. QLPart.SetFloat(t.val); .)
72  | number             (. QLPart.SetNumber(t.val); .)
73  | star               (. QLPart.IsNull = true; .)
74  | myclass<out QLClass>       (. QLPart.QLClass = QLClass; .)
75  | "()"               (. QLPart.IsEmptyList = true; .).
76
77  /*=====================================================*/
78  myclass<out QLClass QLClass>  (. QLClass  = new QLClass();
79                       string name; .)
80  = classname <out name>       (. QLList QLList;
81                       QLClass.ClassName = name; .)
82  list<out QLList>           (. QLClass.QLDirectList = QLList;   .).
83
84  /*=====================================================*/
85  entity<out QLEntity QLEntity> (. QLEntity  = new QLEntity();   .)
86  = entityId             (. var value = t.val; .)
87  equal                (. QLClass QLClass;   .)
88  myclass<out QLClass>
89  semicolonT             (. QLEntity.SetEntityAndClass(value, QLClass);  .).
90
91
92  /*=====================================================*/
93  EXCHANGEFILE           (. QLEntity QLEntity;   .)
```

```
94    = entity<out QLEntity>     (. QLExchangeFile.Add(QLEntity); .)
95    {
96      entity<out QLEntity>     (. QLExchangeFile.Add(QLEntity); .)
97    }.
98
99  END EXCHANGEFILE.
```

Auflistung A.6: Grammatik in EBNFzur Erstellung IFC-Instance-Parser für Coco/R

A.12 Laufzeiten der Musteranfragen, ungekürzt

Operation	Anzahl Input Entitäten	Anzahl Output Entitäten	Laufzeit in Milisekunden
ImportModelGeo_1	0	4213	11085,7
ImportModelRsTree_1	0	4213	10527,7
TypeFilter_2	421040	2287	244,3
Operation	Anzahl Input Entitäten	Anzahl Output Entitäten	Laufzeit in Milisekunden
ImportModelGeo_1	0	4213	12103,90
ImportModelRsTree_1	0	4213	10481,50
Deassociater_2	421040	442	231,20
TypeFilter_3	442	442	0,20
TypeFilter_4	442	442	1,30
AttributeFilter_5	442	442	0,20
Operation	Anzahl Input Entitäten	Anzahl Output Entitäten	Laufzeit in Milisekunden
ImportModelGeo_1	0	4213	11992,10
ImportModelRsTree_1	0	4213	10390,20
Deassociater_2	421085	3065	233,60
TypeFilter_3	3065	2287	3,40
AttributeFilter_4	2287	162	0,00
TaskTimer_5	162	15	0,40
Maximum_6	15	2	0,20
Projector_7	2	2	0,30
Operation	Anzahl Input Entitäten	Anzahl Output Entitäten	Laufzeit in Milisekunden
ImportModelGeo_1	0	4213	12006,40
ImportModelRsTree_1	0	4213	10516,20
TypeFilter_2	421040	54	242,90
TypeFilter_3	421040	2287	223,90
Touches_4	123498	1077	2866,10

Abbildung A.6: Laufzeiten der Musteranfrage 1-4, mit Import und räumlicher Indexierung

A.13 Studentenbefragung: Vergleich von QL4BIM und SQL

Diese Befragung wurde 2015 durchgeführt. Daher unterscheidet sich die gezeigte Sprachsyntax zur der in dieser Arbeit vorgestellten, weiterentwickelten Variante. Dies gilt für die textuelle und die visuelle Notation von QL4BIM.

First Query

"Select all doors which are higher than 200 centimetres and show the OperationType REVOLVING."

Listing 1 First Query in SQL

```
USE TUMcampus;
SELECT * FROM IfcDoor AS Door
WHERE Door.OverallHeight > 200
AND Door.OperationType = "REVOLVING";
```

Please rate your comprehension of query 1 in SQL on a scale from 1 to 10:

1 → I understand absolutely nothing ☐
5 → I understand the major parts of the query ☐
7 → I fully understand the query ☐
10 → I am capable of formulating such a query ☐
_ ← I want to state a value between. (Choose a value from 1-10)

Listing 2 First Query in QL4BIM

```
allObjects = GetModel ("C:\TUMcampus.ifc")
allDoors = TypeFilter (allObjects, "IfcDoor")
highDoors = AttributeFilter (allDoors.OverallHeight > 200)
doors = AttributeFilter (highDoors.OperationType = "
   REVOLVING")
```

Please rate your comprehension of query 1 in QL4BIM on a scale from 1 to 10:

1 → I understand absolutely nothing ☐
5 → I understand the major parts of the query ☐
7 → I fully understand the query ☐
10 → I am capable of formulating such a query ☐
_ ← I want to state a value between. (Choose a value from 1-10)

Second Query

> **"Select every wall and the storey in which the wall is located. Only storeys with an elevation of at least 4 meters should be considered."**

Listing 3 Second Query in SQL

```
USE TUMcampus;
SELECT Wall.*, Storey.* FROM
   IfcRelContainedInSpatialStructure AS RelStructure
JOIN SpatialRelatedElements AS RelatedElements
   ON RelStructure.Id = RelatedElements.Container
JOIN IfcWall AS Wall
   ON RelatedElements.RelatedElement = Wall.Id
JOIN IfcBuildingStorey AS Storey
   ON RelStructure.RelatingStructure = Storey.Id
WHERE Storey.Elevation >= 4.0
```

Please rate your comprehension of query 2 in SQL on a scale from 1 to 10:

1 →	I understand absolutely nothing	☐
5 →	I understand the major parts of the query	☐
7 →	I fully understand the query	☐
10 →	I am capable of formulating such a query	☐
_ ←	I want to state a value between. (Choose a value from 1-10)	

Listing 4 Second Query in QL4BIM

```
allObjects = GetModel("C:\TUMcampus.ifc")
walls = TypeFilter(allObjects, "IfcWall")
allStoreys = TypeFilter(allObjects, "IfcBuildingStorey")
storeys = AttributeFilter(allStoreys.Elevation >= 4.0)
pair[w, s] = RelationResolver(walls, storeys, "
   ContainedInStructure")
```

Please rate your comprehension of query 2 in QL4BIM on a scale from 1 to 10:

1 → I understand absolutely nothing ☐
5 → I understand the major parts of the query ☐
7 → I fully understand the query ☐
10 → I am capable of formulating such a query ☐
_ ← I want to state a value between. (Choose a value from 1-10)

First Query Stated in the Visual Notation of QL4BIM

"Select all doors which are higher than 200 centimetres and show the OperationType REVOLVING."

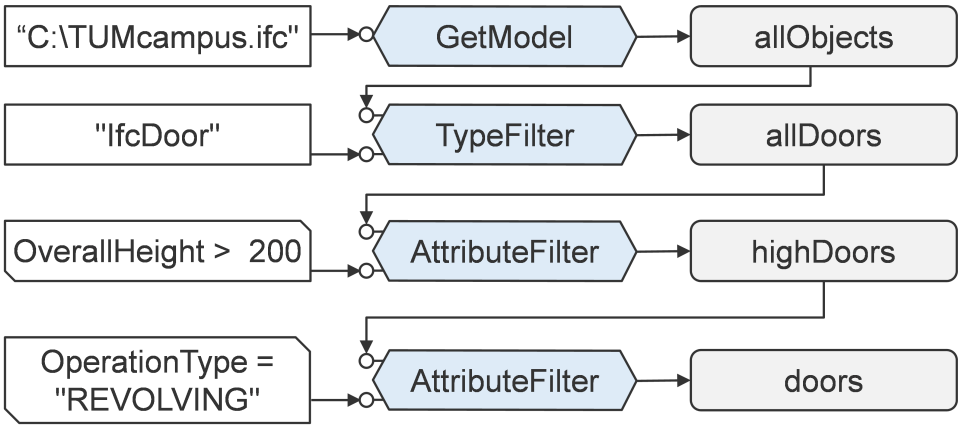


Figure 1: Query 1 in the visual notation of QL4BIM

Please rate your comprehension of query 1 in visual QL4BIM on a scale from 1 to 10:

1 → I understand absolutely nothing ☐
5 → I understand the major parts of the query ☐
7 → I fully understand the query ☐
10 → I am capable of formulating such a query ☐
_ ← I want to state a value between. (Choose a value from 1-10)

Literaturverzeichnis

Abdel-Malek, K., Yang, J., Blackmore, D. & Ken, J. (2006). Swept volumes: fundation, perspectives, and applications. *International Journal of Shape Modeling* 12(01), S. 87–127.

Adachi, Y. (2002a). IFC Model Server Development Project: SECOM Co., Ltd and VTT Building and Construction. http://cic.vtt.fi/projects/ifcsvr/tec/VTT-TEC-ADA-12.pdf.

Adachi, Y. (2002b). Overview of partial model query language: VTT-TEC-ADA-12: SECOM Co., Ltd. / VTT Building and Transport. http://cic.vtt.fi/projects/ifcsvr/tec/VTT-TEC-ADA-12.pdf.

Adachi, Y. (2002c). Technical Overview of IFC Model Server: VTT-TEC-ADA-11: SECOM Co., Ltd. / VTT Building and Transport. http://cic.vtt.fi/projects/ifcsvr/tec/VTT-TEC-ADA-11.pdf.

Adachi, Y. (2015). PSD for IFC4 summary: Property Set Definition. http://www.buildingsmart-tech.org/specifications/pset-releases/psd-for-ifc4/psd-for-ifc4-summary.

Ahmed, K. & Moore, G. (2014). Polaris Management Tool: BrightstarDB 1.9.1 documentation. http://brightstardb.readthedocs.org/en/latest/Using_Polaris/.

Akinci, B., Fischen, M., Levitt, R. & Carlson, R. (2002). Formalization and Automation of Time-Space Conflict Analysis. *Journal of Computing in Civil Engineering* 16(2), S. 124–134.

Albahari, J. & Albahari, B. (2012). *C# 5.0 in a Nutshell: The Definitive Reference*. O'Reilly Media, Inc.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2), S. 123–154.

Anders, B., Mary, F., Ashok, M., Jonathan, M., Marton, N. & Norman, W. (2010). XQuery 1.0 and XPath 2.0 data model: W3C Recommendation 14 December 2010 (revised 7 September 2015). https://www.w3.org/TR/xpath-datamodel/.

Apache Software Foundation (2011). The Apache DB Project. https://db.apache.org/ojb/docu/tutorials/odmg-tutorial.html#Querying+Persistent+Objects.

Armstrong, M. A. (1983). *Basic Topology*. Undergraduate Texts in Mathematics. New York, NY: Springer New York.

Autodesk (1998). DXF Reference. http://www.autodesk.com/techpubs/autocad/acadr14/dxf/.

Bailey, I., Hardwick, M., Laud, A. & Spooner, D. (1996). Overview of the EXPRESS-X Language. In: *Proceedings of the Sixth EXPRESS Users Group Conference, Toronto, Canada, 5q6 October*.

Bao, J., Calvanese, D., Grau, B., Dzbor, M., Fokoue, A., Golbreich, C., Hawke, S., Herman, I., Hoekstra, R., Horrocks, I., Kendall, E., Krötzsch, M., Lutz, C., McGuinness, D., Motik, B., Pan, J., Parsia, B., Schneider, P., Rudolph, S., Ruttenberg, A., Sattler, U., Schneider, M., Smith, M., Wallace, E., Wu, Z. & Zimmermann, A. (2012). OWL 2 Web Ontology Language: Document Overview (Second Edition): W3C Recommendation 11 December 2012. http://www.w3.org/TR/owl2-overview/.

Bartelme, N. (2005). *Geoinformatik: Modelle, Strukturen, Funktionen* (4 Aufl.). Berlin: Springer.

Battle, R. & Kolas, D. (2012). Enabling the geospatial semantic web with parliament and geosparql. *Semantic Web* 3(4), S. 355–370.

Baumgart, B. G. (1972). Winged Edge Polyhedron Representation. Forschungsbericht, Stanford, CA, USA.

Baumgart, B. G. (1975). A polyhedron representation for computer vision. In: *Proceedings of the May 19-22, 1975, national computer conference and exposition*, S. 589–596.

Bayer, R. & McCreight, E. M. (1972). Organization and maintenance of large ordered indexes. *Acta informatica* 1(3), S. 173–189.

Beckmann, N., Kriegel, H.-P., Schneider, R. & Seeger, B. (1990). *The R*-tree: an efficient and robust access method for points and rectangles*, Volume 19. ACM.

Beetz, J., van Berlo, L., de Laat, R. & van den Helm, Pim (2010). Bimserver. org–an Open Source IFC model server. In: *Proceedings of the CIP W78 conference*.

Beetz, J., van Leeuwen, J. & de Vries, B. (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 23(01), S. 89.

Behm, A., Geppert, A. & Dittrich, K. R. (1997). *On the migration of relational schemas and data to object-oriented database systems*. Citeseer.

Bereta, K., Smeros, P. & Koubarakis, M. (2013). Representation and querying of valid time of triples in linked geospatial data. In: *The Semantic Web: Semantics and Big Data*, S. 259–274. Springer.

Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J. & Siméon, J. (2007). XML Path Language (XPath): 2.0: W3C Recommendation 23 January 2007. https://www.w3.org/TR/2007/REC-xpath20-20070123/.

Berners-Lee, T., Hendler, J., Lassila, O. et al. (2001). The semantic web. *Scientific american* 284(5), S. 28–37.

Bernhardsen, T. & Viak, A. (2002). *Geographic information systems* (3 Aufl.). Arendal, Norway: John Wiley & Sons, Inc.

Bill, R. (2010). *Grundlagen der Geoinformationssysteme* (5 Aufl.). Berlin: Wichmann.

Bill, R. & Zehner, M. L. (2001). *Lexikon der Geoinformatik.* Heidelberg: Wichmann.

Bishr, Y. (1998). Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science* 12(4), S. 299–314.

Bizer, C., Heath, T. & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems* 5(3), S. 1–22.

Booch, G. (1994). *Object-oriented analysis and design with applications* (2nd ed Aufl.). The Benjamin/Cummings series in object-oriented software engineering. Redwood City, Calif.: Benjamin/Cummings Pub. Co.

Borrmann, A. (2006). *Extended formal specifications of 3D Spatial Data Types: Technical Report.* Dissertation, Technische Universität München., München.

Borrmann, A. (2007). *Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken.* Dissertation, Technische Universität München, München.

Borrmann, A., Koch, C. & Beetz, J. (Hrsg.) (2015). *Building Information Modeling: Technologische Grundlagen und industrielle Praxis.* SpringerLink : Bücher. Wiesbaden: Springer Vieweg.

Borrmann, A., Kolbe, T. H., Donaubauer, A., Steuer, H., Jubierre, J. R. & Flurl, M. (2015). Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications. *Computer-Aided Civil and Infrastructure Engineering* 30(4), S. 263–281.

Borrmann, A. & Rank, E. (2009a). Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics* 23(1), S. 32–44.

Borrmann, A. & Rank, E. (2009b). Topological analysis of 3D building models using a spatial query language. *Advanced Engineering Informatics* 23(4), S. 370–385.

Borrmann, A. & van Treeck, C. (2006). Ein Algorithmus für die rekursive Abstandbestimmung Spacetree-codierter Rastergeometrien. In: *18. Forum Bauinformatik*, Weimar, Germany.

Braun, A., Tuttas, S., Borrmann, A. & Stilla, U. (2014). Towards automated construction progress monitoring using BIM-based point cloud processing. In: *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*, Vienna, Austria.

Braz, L. M. (1990). Visual syntax diagrams for programming language statements. In: *ACM SIGDOC Asterisk Journal of Computer Documentation*, Volume 14, S. 23–27.

Brill, M. (2014). *Modellierung und kollaboratives Arbeiten mit aktueller BIM-Software am Beispiel des Forschungszentrums TranslaTUM.* Bachelorarbeit, Technische Universität München, München.

Brinkhoff, T. (2013). *Geodatenbanksysteme in Theorie und Praxis: Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial* (3., überarb. und erw. Aufl Aufl.). Heidelberg: Wichmann.

BuildingSMART (2007). Structural Analysis View Summary. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/structural-analysis-view.

BuildingSMART (2009a). Coordination View Version 2.0 Summary: Purpose of Coordination View Version 2.0. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/coordination-view-v2.0.

BuildingSMART (2009b). FM Handover Aquarium. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/fm-handover-aquarium.

BuildingSMART (2010). Space Boundary Summary. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-design-transfer-view.

BuildingSMART (2014a). IFC4 Design Transfer View: Purpose of the IFC4 Design Transfer View. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-design-transfer-view.

BuildingSMART (2014b). IFC4 RV Objective: Workflow support by the IFC4 Reference View. http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-reference-view/ifc4-rv-objective.

BuildingSMART (2014c). ifcXML4 Release summary. http://www.buildingsmart-tech.org/specifications/ifcxml-releases/ifcxml4-release.

BuildingSMART (2015). BuildingSMART - About. http://www.buildingsmart.org/about/.

BuildingSMART (2017a). Pset_WindowCommon: Properties common to the definition of all occurrences of Window. http://www.buildingsmart-tech.org/ifc/IFC4/final/html/psd/Pset_WindowCommon.xml.

BuildingSMART (2017b). Summary of Property Set Releases. http://www.buildingsmart-tech.org/specifications/pset-releases.

buildingSMART International Ltd (2013a). ifcXML4: Full ifcXML schema definition (XSD) for the IFC4 release (released for production). http://www.buildingsmart-tech.org/ifcXML/IFC4/final/ifcXML4.xsd.

buildingSMART International Ltd (2013b). Industry Foundation Classes Release 4 (IFC4): 5.3.3.6 IfcTask. http://www.buildingsmart-tech.org/ifc/IFC4/final/html/link/ifctask.htm.

buildingSMART International Ltd (2013c). Industry Foundation Classes Release 4 (IFC4): EXPRESS file. http://www.buildingsmart-tech.org/downloads/ifc/ifc4/ifc4-express-longform.

Burnett, M. M. & Baker, M. J. (1994). A Classification System for Visual Programming Languages. *Journal of Visual Languages & Computing* 5(3), S. 287–300.

Butterfield, A. & Ekembe Ngondi, G. (Hrsg.) (2016). *A dictionary of computer science* (Seventh edition Aufl.). Oxford Quick Reference. Oxford and New York: Oxford University Press.

Cattell, R. G. G. (Hrsg.) (2000). *The object data standard: ODMG 3.0*. The Morgan Kaufman series in data management systems. San Francisco: Morgan Kaufmann.

Chen, P. P.-S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems* 1(1), S. 9–36.

Chen, Y., Wang, H., Rosen, D. W. & Rossignac, J. (2005). A point-based offsetting method of polygonal meshes. http://www-bcf.usc.edu/~yongchen/Research/MeshOffset.pdf.

Chipman, T., Liebich, T. & Weise, M. (2013). mvdXML: Specification of a standardized format to define and exchangeModel View Definitions with Exchange Requirements and Validation Rules. http://www.buildingsmart-tech.org/downloads/accompanying-documents/formats/mvdxml-documentation/mvdxml-schema-documentation-v1-1-draft/at_download/file.

Clemen, C. (2010). *Ein geometrisch-topologisches Informationsmodell für die Erfassung und Validierung von flächenparametrisierten 3d-Gebäudemodellen*, Volume 647 of *Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften Reihe C, Dissertationen*. München: Verl. der Bayerischen Akad. der Wiss.

Clementini, E. & Di Felice, P. (1995). A comparison of methods for representing topological relationships. *Information Sciences-Applications* 3(3), S. 149–178.

Clementini, E. & Di Felice, P. (1996). A model for representing topological relationships between complex geometric features in spatial databases. *Information sciences* 90(1), S. 121–136.

Clementini, E. & Di Felice, P. (1997). Approximate topological relations. *International Journal of Approximate Reasoning* 16(2), S. 173 – 204.

Clementini, E., Di Felice, P. & van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In: *Advances in Spatial Databases*, S. 277–295.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)* 4(4), S. 397–434.

Codd, E. F. (1990). *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc.

Copeland, G. (1980). What if Mass Storage Were Free? *SIGIR Forum* 15(2), S. 1–7.

Cover, R. (2005). Cover Pages: XML Applications and Initiatives. http://xml.coverpages.org/xmlApplications.html.

Cyganiak, R., Wood, D. & Lanthaler, M. (2014). RDF 1.1 concepts and abstract syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

Daum, S. & Borrmann, A. (2013). Definition and Implementation of Temporal Operators for a 4D Query Language. In: *Proc. of the ASCE International Workshop on Computing in Civil Engineering*.

Daum, S. & Borrmann, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. *Advanced Engineering Informatics* 28(4), S. 272–286.

de Laat, R. & van Berlo, L. (2011). Integration of BIM and GIS: The development of the CityGML GeoBIM extension. In: *Advances in 3D Geo-Information Sciences*, S. 211–225. Springer.

Dobesova, Z. (2011). Visual programming language in geographic information systems. In: *Proceedings of the 2nd international conference on Applied informatics and computing theory*, S. 276–280.

Donkers, S. (2013). *Automatic generation of CityGML LoD3 building models from IFC models*. Dissertation, TU Delft, Delft University of Technology.

Dori, G., Borrmann, A., Szczesny, K., Hamm, M. & König, M. (2012). Combining forward and backward process simulation for generating and analysing construction schedules. In: *Proc. of the 14th Int. Conf. on Computing in Civil and Building Engineering*, Moscow, Russia.

Dyreson, C., Grandi, F., Käfer, W., Kline, N., Lorentzos, N., Mitsopoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B. et al. (1994). A consensus glossary of temporal database concepts. *ACM Sigmod Record* 23(1), S. 52–64.

Eastman, C., Teicholz, P., Sacks, R. & Liston, K. (2008). *BIM Handbook.* Hoboken, NJ, USA: John Wiley & Sons, Inc.

Eberly, D. (1999). Distance between point and triangle in 3D: Geometric Tools, LLC: http://www.geometrictools.com/. https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf.

Eberly, D. (2001). Intersection of convex objects: The method of separating axes. www.magic-software.com.

Egenhofer, M. & Franzosa, R. (1991). Point-Set Topological Spatial Relations. *Int. Journal of Geographical Information Systems* 5(2), S. 161–174.

Egenhofer, M. & Herring, J. (1992). A Mathematical Framework for the Definition of Topological Relationships. In: P. Bresnahan, E. Corwin, & D. Cowen (Hrsg.), *Proc. of the 4th Int. Symp. on Spatial Data Handling.* Charleston, S.C., USA.

Egenhofer, M. & Herring, J. (1994). Categorizing binary topological relations between regions, lines and points in geographic databases, the 9-intersection: Formalism and its Use for Natural language Spatial Predicates. *Santa Barbara CA National Center for Geographic Information and Analysis Technical Report*, S. 94–1.

El-Mekawy, M. (01.01.2010). *Integrating BIM and GIS for 3D city modelling: The case of IFC and CityGML.* Dissertation, KTH.

El-Mekawy, M., Östman, A. & Hijazi, I. (2012). A unified building model for 3D urban GIS. *ISPRS International Journal of Geo-Information* 1(2), S. 120–145.

El-Mekawy, M., Östman, A. & Shahzad, K. (2011). Towards interoperating cityGML and IFC building models: a unified model based approach. In: *Advances in 3D Geo-Information Sciences*, S. 73–93. Springer.

Elmasri, R., Navathe, S. B. & Navathe, S. (2002). *Grundlagen von Datenbanksystemen* (3., überarb. Aufl. Aufl.). Informatik : Datenbanken. München: Pearson Studium.

Enge, F. (2010). *Muster in Prozessen der Bauablaufplanung: Ein Branch-and-Bound-Verfahren zur Mustererkennung in Planungs- und Ausführungsprozessen*, Volume 6 of *Heftreihe des Instituts für Bauingenieurwesen.* Aachen: Shaker.

Fallside, D. C. & Walmsley, P. (2004). XML schema part 0: primer second edition: W3C Recommendation 28 October 2004. https://www.w3.org/TR/xmlschema-0/.

Filip, D. J. & Ball, T. W. (1989). Procedurally representing lofted surfaces. *IEEE Computer Graphics and Applications* 9(6), S. 27–33.

Finkel, R. A. & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica* 4(1), S. 1–9.

Fischer, P. & Hofer, P. (2011). *Lexikon der Informatik* (15., überarb. Aufl. Aufl.). Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Florescu, D., Hillery, C., Kossmann, D., Lucas, P., Riccardi, F., Westmann, T., Carey, M. J., Sundararajan, A. & Agrawal, G. (2003). The BEA/XQRL Streaming XQuery Processor. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, S. 997–1008. VLDB Endowment.

FME (2016). Writing CityGML from FME. http://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_ReadersWriters/citygml/Tutorial/writing_citygml_from_fme.htm.

Forster, O. (2013). *Analysis 2: Differentialrechnung im IRn, gewöhnliche Differentialgleichungen* (10th ed Aufl.). Grundkurs Mathematik. Dordrecht: Springer.

Fowler, J. (1995). *STEP for data management, exchange and sharing.* Twickenham, UK: Ta Technology Appraisals Ltd.

Frank, A. U. (1996). Qualitative spatial reasoning: Cardinal directions as an example. *International Journal of Geographical Information Science* 10(3), S. 269–290.

Gaal, S. A. (1964). *Point set topology*, Volume 16 of *Pure and applied mathematics a series of monographs and textbooks.* New York: Academic Press.

Gabbrielli, M. & Martini, S. (2010). *Programming Languages: Principles and Paradigms.* Undergraduate Topics in Computer Science. London: Springer-Verlag London Limited.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader).* Pearson Education.

Gao, S., Sperberg-McQueen, C. M., Thompson, H. S., Mendelsohn, N., Beech, D. & Maloney, M. (2012). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures: W3C Recommendation 5 April 2012. https://www.w3.org/TR/xmlschema11-1/.

Gerschwitz, A. (2011). Die dritte Dimension im ALKIS: Landesamt für Vermessung und Geoinformation Schleswig-Holstein. http://www.dvw.de/sites/default/files/arbeitskreis/2/anhang/2011/09-Gerschwitz-3D-ALKIS.pdf.

Gijezen, S., Hartmann, T., Veenvliet, K., Hendriks, H. & Buursema, N. (2010). *Organizing 3D building information models with the help of work breakdown structures to improve the clash detection process.* Final repor, University of Twente, Enschede, The Netherlands.

Gottschalk, S., Lin, M. C. & Manocha, D. (1996). OBBTree: A hierarchical structure for rapid interference detection. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, S. 171–180.

Goyal, R. K. (2000). *Similarity assessment for cardinal directions between extended spatial objects.* Dissertation, Citeseer.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition* 5(2), S. 199–220.

Grün, C. (2015). http://basex.org/about-us/.

Gu, P. & Chan, K. (1995). Product modelling using step. *Computer-Aided Design* 27(3), S. 163–179.

Guesgen, H. W. (1989). *Spatial reasoning based on Allen's temporal logic.* International Computer Science Institute Berkeley.

Güting, R. H. (1988). *Geo-relational algebra: A model and query language for geometric database systems.* Springer.

Güting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal—The International Journal on Very Large Data Bases* 3(4), S. 357–399.

Guttman, A. (1984). R-trees: A Dynamic Index Structure for Spatial Searching. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, New York, NY, USA, S. 47–57. ACM.

Harold, E. R., Means, W. S. & Udemadu, K. (2004). *XML in a Nutshell*, Volume 8. O'reilly Sebastopol, CA.

Heath, T. & Bizer, C. (2011). *Linked Data: Evolving the web into a global data space.* Synthesis lectures on the semantic web theory and technology. San Rafael: Morgan & Claypool.

Herring (2011a). OpenGIS\textregistered Implementation Standard for Geographic information-Simple feature access-Part 1: Common architecture.,(OGC 06-103r4). *OpenGIS\textregistered Implementation Standard* 28.

Herring, J. R. (2011b). Implementation Standard for Geographicinformation - Simple feature access - Part 1: Common architecture: - OGC 06-103r4: 1.2.1. http://www.opengeospatial.org/standards/sfa.

Hofstadler, C. (2007). *Bauablaufplanung und Logistik im Baubetrieb* (1. Aufl Aufl.). Berlin: Springer.

Horrocks, I., Kutz, O. & Sattler, U. (2006). The Even More Irresistible SROIQ. *Kr* 6, S. 57–67.

Huang, C.-W. & Shih, T.-Y. (1997). On the complexity of point-in-polygon algorithms. *Computers & Geosciences* 23(1), S. 109–118.

Huhnt, W. (2004). Progress measurement in planning processes on the base of process models. In: *Xth International Conference on Computing in Civil and Building Engineering*, S. 02–04.

Hunt, K. P. (1979). An introduction to structured programming. *Behavior Research Methods & Instrumentation* 11(2), S. 229–233.

Hunter, G. M. & Steiglitz, K. (1979). Operations on images using quad trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 1(2), S. 145–153.

Islam, M. S., Kuzu, M. & Kantarcioglu, M. (2015). A Dynamic Approach to Detect Anomalous Queries on Relational Databases. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, New York, NY, USA, S. 245–252. ACM.

ISO (01.10.2007a). Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas. http://www.iso.org/iso/catalogue_detail.htm?csnumber=40646.

ISO (1992). ISO 10303-42:1992 Industrial automation systems and integration – Product data representation and exchange – Part 42: Integrated generic resource: Geometric and topological representation Exchange -Part 1: Overview and Fundamental Principles. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=37846.

ISO (1994). ISO 10303-1:1994, Industrial Automation Systems andIntegration - Product Data Representation and Exchange -Part 1: Overview and Fundamental Principles. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=20579.

ISO (2002a). Industrial automation systems and integration – Product data representation and exchange – methods: Clear text encoding of the exchange structure: Part 21: Implementation. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33713.

ISO (2002b). ISO 19108:2002 Geographic information - Temporal schema. https://www.iso.org/standard/26013.html.

ISO (2007b). ISO 19111:2007: Geographic information – Spatial referencing by coordinates: ISO/TC 211 Geographic information/Geomatics. https://www.iso.org/standard/41126.html.

ISO (2010). Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option: OGC 06-104r4. http://www.opengeospatial.org/standards/sfs.

ISO (2014). ISO 19101-1:2014 Geographic information: Geographic information – Reference model – Part 1: Fundamentals: : ISO/TC 211 Geographic information/Geomatics. https://www.iso.org/standard/59164.html.

ISO (2016a). Information technology – Database languages – SQL multimedia and application packages – Part 3: Spatial. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=60343.

ISO (2016b). International Organization for Standardization. ISO/IEC:2016: Information technology – Database languages – SQL, 2016. https://www.iso.org/standard/63555.html.

ISO 19107 (2003). ISO 19107:2003: Geographic information – Spatial schema: Geographic information – Spatial schema. https://www.iso.org/standard/26012.html.

Itzik, B.-G., Dejan, S., Adam, M. & Kevin, F. (2015). *T-SQL querying.* Redmond, WA: Microsoft.

Jaffar, J. & Lassez, J.-L. (1987). Constraint Logic Programming. In: *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, New York, NY, USA, S. 111–119. ACM.

Johnston, W. M., Hanna, J. R. P. & Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Computing Surveys* 36(1), S. 1–34.

Jones, M. W. (1995). *3D distance from a point to a triangle: CSR-5.* Technical report, University of Wales Swansea, Swansea, UK.

JTC (1996). Information technology — Syntactic metalanguage — Extended BNF: ISO/IEC 14977. https://www.iso.org/obp/ui/#iso:std:iso-iec:14977:ed-1:v1:en.

Kappas, M. (2011). *GIS* (1. Aufl Aufl.), Volume 8 of *Das geographische Seminar*. Braunschweig [u.a.]: Westermann.

Kemper, A. (2004). *Datenbanksysteme: Eine Einführung* (5., aktual. und erw. Aufl Aufl.). München: Oldenbourg.

Kim, S.-J., Lee, D.-Y. & Yang, M.-Y. (2004). Offset triangular mesh using the multiple normal vectors of a vertex. *Computer-Aided Design and Applications* 1(1-4), S. 285–291.

Klein, L. (2010). JSDAI: Java Standard Data Access Interface. http://www.jsdai.net/.

Klug, A. (1982). Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM (JACM)* 29(3), S. 699–717.

Knauff, M., Rauh, R. & Renz, J. (1997). A cognitive assessment of topological spatial relations: Results from an empirical investigation. In: *Spatial Information Theory A Theoretical Basis for GIS*, S. 193–206. Springer.

Kolbe, T. H. (2009). Representing and exchanging 3D city models with CityGML. In: *3D geo-information sciences*, S. 15–31. Springer.

Kolbe, T. H. & Gröger, G. (2004). Towards unified 3D city models. In: *Proc. of the ISPRS Comm. IV Joint Workshop on Challenges in Geospatial Analysis.*

König, M., Beißert, U., Steinhauer, D. & Bargstädt, H.-J. (2007). Constraint-based simulation of outfitting processes in shipbuilding and civil engineering. In: *Proceedings of the 6th EUROSIM Congress on Modeling and Simulation.*

Kowalski, R. (1979). Algorithm= logic+ control. *Communications of the ACM* 22(7), S. 424–436.

Kulkarni, K. & Michels, J.-E. (2012). Temporal features in SQL: 2011. *ACM Sigmod Record* 41(3), S. 34–43.

Laakso, M. & Kiviniemi, A. O. (2012). The IFC standard: A review of History, development, and standardization, Information Technology. *ITcon* 17(9), S. 134–161.

Laidlaw, D. H., Trumbore, W. B. & Hughes, J. F. (1986). Constructive solid geometry for polyhedral objects. In: *ACM SIGGRAPH computer graphics*, Volume 20, S. 161–170.

Lange, N. d. (2013). *Geoinformatik in Theorie und Praxis* (3 Aufl.). Berlin: Springer Spektrum.

Langran, G. & Chrisman, N. R. (1988). A framework for temporal geographic information. *Cartographica: The International Journal for Geographic Information and Geovisualization* 25(3), S. 1–14.

Lee, J. K. (2010). *Building environment rule and analysis (BERA) language and its application for evaluating building circulation and spatial program.* Dissertation - doctor of philosophy, Georgia Institute of Technology, Georgia. USA.

Leffler, J. & Savage, R. (2017). BNF Grammars for SQL-92, SQL-99 and SQL-2003. https://github.com/ronsavage/SQL.

Liebich, T. (2001). XML schema language binding of EXPRESS for ifcXML. *International Alliance for Interoperability*.

Liebich, T. (2013). IFC4 specification. http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/index.htm.

Liebich, T., Adachi, Y., Forester, J., Hyvarinen, J., Richter, S., Chipman, T., Matthias, W. & Jeffrey, W. (2013). Industry Foundation Classes IFC4 Official Release: Core data schemas. http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/chapter-5.htm.

Lieu, D. K. & Sorby, S. A. (2009). *Visualization, modeling, and graphics for engineering design* (1st ed. Aufl.). Clifton Park, NY: Delmar Cengage Learning.

Lima, C., El-Diraby, T. & Stephens, J. (2005). Ontology-based optimization of knowledge management in e-construction. *ITcon* 10(2005), S. 305–327.

Lindholm, T. (2004). A three-way merge for XML documents. In: E. V. Munson & J.-Y. Vion-Dury (Hrsg.), *the 2004 ACM symposium*, S. 1.

Lother, G. (2003). *Konzeptionelle Aspekte eines landesweiten Fachgeoinformationssystems für die Bestandsdokumentation forstlicher Geodaten.* Dissertation, Technische Universität München, München.

Lother, G. (2007). Skript zu Vorlesung GeoInformatik 2: Hochschule für angewandte Wissenschaften München.

Louden, K. C. & Lambert, K. A. (2012). *Programming languages: Principles and practice* (3rd ed. Aufl.). Boston, MA: Course Technology/Cengage Learning.

MacLennan, B. J. (1983). Overview of relational programming. *ACM SIGPLAN Notices* 18(3), S. 36.

Maes, P. (1987). Concepts and experiments in computational reflection. In: N. Meyrowitz (Hrsg.), *Conference proceedings*, S. 147–155.

Malosio, M., Pedrocchi, N. & Tosatti, L. M. (2009). Algorithm to offset and smooth tessellated surfaces. *Computer-Aided Design and Applications* 6(3), S. 351–363.

Manthey, R. (2002). Skript zur Vorlesung Informationssysteme: Relationale Anfragesprachen. http://www.iai.uni-bonn.de/III/lehre/vorlesungen/Informationssysteme/WS02/folien/DB5b-1.pdf.

Mäntylä, M. (1988). *An introduction to solid modeling*, Volume 13 of *Principles of computer science series.* Rockville, Md.: Computer Science Press.

Masuda, H. (1993). Topological operators and Boolean operations for complex-based nonmanifold geometric models. *Computer-Aided Design* 25(2), S. 119–129.

Mazairac, W. & Beetz, J. (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics* 27(4), S. 444–456.

McKinney, K. & Fischer, M. (1998). Generating, evaluating and visualizing construction schedules with CAD tools. *Automation in Construction* 7, S. 433–447.

Meijer, E., Beckman, B. & Bierman, G. (2006). LINQ: Reconciling Object, Relations and XML in the .NET Framework. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, New York, NY, USA, S. 706. ACM.

Metthews, A., Blackwell, B., Barker, M., Fraser, N., Jackson, P., Thomas, A., Shah, S., Rawlinson, S. & Boniface, T. (2011). Strategy Paper for the Government Construction Client Group. http://www.bimtaskgroup.org/wp-content/uploads/2012/03/BIS-BIM-strategy-Report.pdf.

Michaelson, G. (2013). *An Introduction to Functional Programming Through Lambda Calculus*. Dover Books on Mathematics. Newburyport: Dover Publications.

microsoft (2015). Standard Query Operators Overview. https://msdn.microsoft.com/en-us/library/bb397896.aspx.

Microsoft Developer Network (2014). Onlinedokumentation für SQL Server 2014. https://msdn.microsoft.com/library/ms130214(v=sql.120).aspx.

Möller, T. (1997). A fast triangle-triangle intersection test. *Journal of graphics tools* 2(2), S. 25–30.

Monedero, J. (2000). Parametric design: A review and some experiences. *Automation in Construction* 9(4), S. 369–377.

Moon, H., Dawood, N. & Kang, L. (2014). Development of workspace conflict visualization system using 4D object of work schedule. *Advanced Engineering Informatics* 28(1), S. 50–65.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1(1), S. 97–123.

Nagel, C., Stadler, A. & Kolbe, T. H. (2009). Conceptual Requirements for the Automatic Reconstruction of Building Information Models from Uninterpreted 3D Models. In: T. H. Kolbe, R. Zhang, & S. Zlatanova (Hrsg.), *Proceedings of the Academic Track of the Geoweb 2009 - 3D Cityscapes Conference in Vancouver,Canada, 27-31 July 2009*, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, S. 46–53. ISPRS.

Nash, F., Du Preez, A. & de Wet, C. (2009). *FCS Computer Programming L4*. Pearson South Africa.

Nitithamyong, P. & Skibniewski, M. J. (2006). Success/Failure Factors and Performance Measures of Web-Based Construction Project Management Systems: Professionals' Viewpoint. *Journal of Construction Engineering and Management* 132(1), S. 80–87.

Nour, M. & Beucke, K. (2008). An open platform for processing IFC model versions. *Tsinghua Science and Technology* 13(S1), S. 126–131.

Oberbichler, T., Reith, C. & Marcinonis, D. (2015). *Arbeitsbericht zur Lehrveranstaltung Building Information Modeling.* Dissertation, Lehrstuhl für Computergestützte Modellierung und Simulation, TUM, München.

OGC (2011). Unified modeling language (OMG UML), Superstructure. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/.

OGC (2012). OGC City Geography Markup Language (CityGML) Encoding Standard. https://portal.opengeospatial.org/files/?artifact_id=47842.

OGC (2016). OGC Standards. http://www.opengeospatial.org/docs/is.

Oppel, A. J. (2010). *Data modeling: A beginner's guide.* New York: McGraw-Hill.

Osada, R., Funkhouser, T., Chazelle, B. & Dobkin, D. (2002). Shape distributions. *ACM Transactions on Graphics (TOG)* 21(4), S. 807–832.

Ott, T. & Swiaczny, F. (2001). *Time-integrative geographic information systems: Management and analysis of spatio-temporal data.* Berlin and New York: Springer.

Oxford Brookes University (2002). XML Primer. www.w3c.it/education/2012/upra/documents/xmlprimer.pdf.

Parr, T. (2010). *Language implementation patterns: Create your own domain-specific and general programming languages.* The pragmatic programmers. Raleigh, N.C.: Pragmatic Bookshelf.

Partsch, H. A. (2010). *Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme.* EXamen.press. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Pauwels, P. & Roxin, A. (2016). SimpleBIM: From full ifcOWL graphs to simplified building graphs. In: *11th European Conference on Product and Process Modelling (ECPPM)*, S. 11–18.

Pauwels, P. & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction* 63, S. 100–133.

Pelekis, N., Theodoulidis, B., Kopanakis, I. & Theodoridis, Y. (2004). Literature review of spatio-temporal database models. *The Knowledge Engineering Review* 19(03), S. 235–274.

Perry, M. & Herring, J. (2012). OGC GeoSPARQL-A geographic query language for RDF data: OGC 11-052r4. http://www.opengis.net/doc/IS/geosparql/1.0.

Peuquet, D. J. (1999). Time in GIS and geographical databases. *Geographical information systems* 1, S. 91–102.

Peyton Jones, S. & Wadler, P. (1993). Imperative functional programming. In: *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, S. 71–84.

PostGIS (22.05.2016). PostGIS 2.2.3 Dev Manual: SVN Revision (14900). http://postgis.net/docs/PostGIS_Special_Functions_Index.html#PostGIS_3D_Functions.

Pratt, M. J. (2001, March). Introduction to ISO 10303&Mdash;the STEP Standard for Product Data Exchange. *J. Comput. Info. Sci. Eng.* 1(1), S. 102–103.

Reed, K. (1991). Initial Graphics Exchange Specification, National Computer Graphics Association.

Reinhardt, W. (2011). Vorlesung Geoinformatik II: Kapitel 3 Spatial Schema. https://www.unibw.de/inf4/professuren/geoinformatik/lehre/skripten/skripte/skripten_ht_11/standards-c5-spatialschema_neu_171011.pdf.

Renolen, A. (1997). Conceptual modelling and spatiotemporal information systems: how to model the real world. In: *6th SCANDINAVIAN RESEARCH CONFERENCE ON GIS (SCANGIS'97), Estocolmo.*

Requicha, A. A. G. (1980). Representations of rigid solid objects. In: J. Encarnação (Hrsg.), *Computer aided design modelling, systems engineering, CAD-systems*, Volume 89 of *Lecture Notes in Computer Science*, S. 1–78. Berlin: Springer.

Richard, S., Karlshoej, J. & Davis, D. (2012). An Integrated Process for Delivering IFC Based Data Exchange. http://bips.dk/files/integrated_idm-mvd_processformats_14_0.pdf.

Ritter, F., Preidel, C. & Singer, D. (2015). Visuelle Programmiersprachen im Bauwesen - Stand der Technik und aktuelle Entwicklungen. In: *Proceedings of the 27th Forum Bauinformatik*, Aachen, Germany.

Rossignac, J. R. & Requicha, A. A. G. (1986). Offsetting operations in solid modelling. *Computer Aided Geometric Design* 3(2), S. 129–148.

Roussopoulos, N., Kelley, S. & Vincent, F. (1995). Nearest neighbor queries. In: *ACM sigmod record*, Volume 24, S. 71–79.

Sacks, R., Eastman, C. M. & Lee, G. (2004). Parametric 3D modeling in building construction with examples from precast concrete. *Automation in Construction* 13(3), S. 291–312.

Sacks, R., Koskela, L., Dave, B. A. & Owen, R. (2010). The interaction of lean and building information modeling in construction. *Journal of Construction Engineering and Management* 136(9), S. 968–980.

Samet, H. & Webber, R. E. (1985). Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics (TOG)* 4(3), S. 182–222.

Schenck, D. & Wilson, P. R. (1994). *Information modeling: the EXPRESS way*, Volume 126. Oxford University Press New York.

Schiffer, S. (1998). *Visuelle Programmierung: Grundlagen und Einsatzmöglichkeiten*. Bonn: Addison-Wesley.

Scott, W. A. (2000). Mapping Objects to Relational Databases. https://www.ibm.com/developerworks/library/ws-mapping-to-rdb/.

Shadbolt, N., Hall, W. & Berners-Lee, T. (2006). The semantic web revisited. *Intelligent Systems, IEEE* 21(3), S. 96–101.

Shaffer, C. A. (2012). *Data Structures and Algorithm Analysis in Java, Third Edition* (3rd ed. Aufl.). Dover Books on Computer Science. Newburyport: Dover Publications.

Shi, W., Fisher, P. F. & Goodchild, M. F. (2002). *Spatial data quality*. London and New York: Taylor & Francis.

Shilane, P., Min, P., Kazhdan, M. & Funkhouser, T. (2004). The princeton shape benchmark. In: *Shape modeling applications, 2004. Proceedings*, S. 167–178.

Shreiner, D. (2013). *OpenGL programming guide: The official guide to learning OpenGL, Version 4.3* (8. ed., 1. print Aufl.). OpenGL series : Graphics programming. Upper Saddle River NJ u.a.: Addison-Wesley.

Simon, C., Paul, D., Ron, L., Clemens, P. & Arliss, W. (2004). OpenGIS® Geography Markup Language (GML) Implementation Specification: OpenGIS® Recommendation Paper: ISO/CD 19136.

Sipser, M. (2003). *Introduction to the theory of computation* (10. Aufl.). Boston, Mass.: PWS Publ.

Smith, R. (1988). GNU diff3: Version 2.8. 1. https://www.esrl.noaa.gov/gmd/dv/hats/cats/stations/qnxman/diff3.html.

Snodgrass, R. T. (1986). Temporal databases. *IEEE Computer* 19, S. 35–42.

Stadler, A., Nagel, C., König, G. & Kolbe, T. H. (2009). Making interoperability persistent: A 3D geo database based on CityGML. In: *3D Geo-Information Sciences*, S. 175–192. Springer.

Stanford University (2008). Data Modelling Using EXPRESS-G for IFC Development. http://plm.ecp.fr/ap242_documents/EXP-G%20-%20Guide%20STANFORD.pdf.

Stroustrup, B. (03.10.2012). Bjarne Stroustrup's C++ Glossary. http://www.stroustrup.com/glossary.html.

Swisher, J. R., Hyden, P. D., Jacobson, S. H. & Schruben, L. W. (2000). A survey of simulation optimization techniques and procedures. In: *Simulation Conference, 2000. Proceedings. Winter*, Volume 1, S. 119–128.

Tauscher, E. & Smarsly, K. (2016). Generic BIM queries based on the IFC object model using graph theory. In: *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering (ICCCBE).*

TC184 (2004). Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual: ISO Standard 10303-11: ISO/TC 184/SC 4 Industrial data. https://www.iso.org/standard/38047.html.

Terkaj, W. & Šojić, A. (2015). Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology. *Automation in Construction* 57, S. 188–201.

Tom, H. & Roswell, C. (2009). Standards Guide: ISO/TC 211 Geographic Information/Geomatics. http://www.isotc211.org/Outreach/ISO_TC_211_Standards_Guide.pdf.

Torgersson, O. (1996). A note on declarative programming paradigms and the future of definitional programming. *Das Winteroete* 96(1996), S. 13.

Torma, S. (2013). Semantic Linking of Building Information Models. In: *IEEE Seventh International Conference on Semantic Computing (ICSC)*, S. 412–419.

Törmä, S., VuHoang, N. & Pauwels, P. (2014). IFC-to-RDF conversion tool. http://linkedbuildingdata.net/tools/tool-ifc-to-rdf-conversion-tool/.

Trybulec, A., Matuszewski, R. & Zalewska, A. (2007). *From insight to proof: Festschrift in honour of Andrzej Trybulec*, Volume 10(23) of *Studies in logic, grammar, and rhetoric.* Białystok: University of Białystok.

Tulke, J. (2010). *Kollaborative Terminplanung auf Basis von Bauwerksinformationsmodellen.* Dissertation, Bauhaus-Universität Weimar, Weimar.

Tulke, J. & Hanff, J. (2007). 4D construction sequence planning–new process and data model. In: *Proceedings of CIB-W78 24th International Conference on Information Technology in Construction, Maribor, Slovenia*, S. 79–84.

Tulke, J., Nour, M. & Beucke, K. (2008a). Decomposition of BIM objects for scheduling and 4D simulation. In: *Proceedings of the 7th European Conference on Product and Process Modelling in the Building and related Industries*, S. 653–660.

Tulke, J., Nour, M. & Beucke, K. (2008b). A Dynamic Framework for Construction Scheduling based on BIM using IFC. In: *Proc. 17th Congress of IABSE*, S. 17–19.

Van Roy, P. (2007). The principal programming paradigms: v1.03. https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng101.pdf.

Vilain, M. B. (1982). A System for Reasoning About Time. In: *AAAI*, S. 197–201.

W3C (2001a). An Introduction to Multilingual Web Addresses. http://www.w3.org/International/articles/idn-and-iri/.

W3C (2001b). URIs, URLs, and URNs: Clarifications and Recommendations 1.0: Report from the joint W3C/IETF URI Planning Interest Group: W3C Note 21 September 2001. http://www.w3.org/TR/uri-clarification/.

W3C (2008). SPARQL Query Language for RDF: W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query.

W3C (2013). SPARQL 1.1 Overview: W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-overview/.

W3C (2014a). RDF 1.1 N-Triples: A line-based syntax for an RDF graph: W3C Recommendation 25 February 2014. http://www.w3.org/TR/n-triples/.

W3C (2014b). RDF 1.1 Turtle: Terse RDF Triple Language: W3C Recommendation 25 February 2014. http://www.w3.org/TR/turtle/.

W3C (2014c). RDF Schema 1.1: W3C Recommendation 25 February 2014. http://www.w3.org/TR/rdf-schema.

W3C (2014d). XQuery 3.0: An XML Query Language.

Wagenknecht, C. (2004). *Programmierparadigmen*. Wiesbaden: Vieweg+Teubner Verlag.

Walmsley, P. (2007). *XQuery*. Farnham, Calif.: O'Reilly.

Weise, M. (2015). Repository used by the mvdXML group to support mvdXML specification and implementation: mvdXMLv1-1_example1.xml. https://github.com/BuildingSMART/mvdXML/tree/master/mvdXML1.1/xml.

Wienholt, N. (2004). *Maximizing .NET performance*. Berkeley, Calif.: Apress.

Wirfs-Brock, A. & Terlson, B. (2009). ECMAScript® 2017 Language Specification.

Wix, J. (2007). Quick Guide Business Process Modeling Notation (BPMN): IDM Methodology : Process Mapping. http://iug.buildingsmart.org/idms/methods-and-guides/QuickGuideToBPMN.pdf/at_download/file.

Wix, J. & Karlshoej, J. (2010). Information delivery manual: Guide to components and development methods. http://iug.buildingsmart.org/idms/development/IDMC_004_1_2.pdf.

Worboys, M. F. (1994). A unified model for spatial and temporal information. *The Computer Journal* 37(1), S. 26–34.

Wöß, A., Löberbauer, M. & Mössenböck, H. (2003). LL (1) conflict resolution in a recursive descent compiler generator. In: *Joint Modular Languages Conference*, S. 192–201.

Wülfing, A., Windisch, R. & Scherer, R. (2015). A visual BIM query language. In: A. Mahdavi, B. Martens, & R. Scherer (Hrsg.), *eWork and eBusiness in architecture, engineering and construction*, A Balkema book, S. 157–164. Boca Raton, Fla.: CRC press.

Wuu, G. & Dayal, U. (2-3 Feb. 1992). A uniform model for temporal object-oriented databases. In: *Eighth International Conference on Data Engineering*, S. 584–593.

Zhou, M., Cao, J. & Jiang, G. (2014). Design and implementation of data transformation scheme between STEP and XML in CIMS environment. In: Wenli-Yao, H.-C. Liu, & W.-P. Sung (Hrsg.), *Information technology and computer application engineering*, S. 277–280. Boca Raton: CRC press.

Zicari, R. V. (2000). ODMG Standards. http://www.odbms.org/odmg-standard/.

Zimmermann, A. (2012). *Basismodelle der Geoinformatik: Strukturen, Algorithmen und Programmierbeispiele in Java ; mit 48 Tabellen und 42 Listings*. München: Hanser.

Zülch, G. & Stock, P. (Hrsg.) (2010). *Integrationspakete der Simulation: Technik, Organisation und Personal*. Karlsruhe: KIT Scientific Publishing.