Lehrstuhl für Computergestütze Modellierung und Simulation
TUM School of Engineering and Design
Technische Universität München

# Graphbasierter Ansatz zur Analyse von BIM-Raumbeziehungen

Wissenschaftliche Arbeit zur Erlangung des Grades

**Master of Science (M.Sc.)**

an der TUM School of Engineering and Design
der Technischen Universität München.

**Betreut von** Prof. Dr.-Ing. André Borrmann
Dipl.-Ing. Ina Heise
Lehrstuhl für Computergestütze Modellierung und Simulation

**Eingereicht von** Ludwig Englert

**Eingereicht am** 15. Juli 2024

## Abstract

There are many formats for modelling and depicting buildings in such a way that computers can also process and check these models completely or at least partially. However, these analyses are usually limited to what has actually been modelled, as the machine lacks the ability to interpret things and draw conclusions. This is where the principle of *Linked Data* comes in and is utilised in this work, to identify potential problems with the building services or other relations between rooms at an early design stage, even though not a single pipe or cable has been explicitly modelled. For this purpose, an ontology is designed that makes it possible to check whether a service line or another relations between two rooms is generally possible. The ontology is based on the *Building Topology Ontology* and is compatible with *ifcOWL* and the other *LBD* ontologies. Competency questions are defined and SPARQL queries and examples are used to show that the ontology fulfils them. In addition, solutions are developed to transfer the corresponding necessary information from IFC files into an RDF graph.

## Zusammenfassung

Es gibt bereits viele Formate und Wege, um Gebäude so zu modellieren und abzubilden, dass auch Computer diese Modelle vollständig oder zumindest teilweise verarbeiten und prüfen können. Diese Analysen beschränken sich aber meist auf das, was tatsächlich modelliert wurde, da der Maschine die Möglichkeit fehlt, Dinge zu interpretieren und Schlussfolgerungen zu ziehen. In dieser Arbeit wird deshalb das Prinzip *Linked Data* genutzt, um bereits in frühen Leistungsphasen mögliche Probleme mit der Gebäudetechnik oder anderen Verknüpfungen zwischen Räumen zu erkennen, obwohl noch keine einzige Leitung explizit modelliert wurde. Dafür wird eine Ontologie entworfen, mit der es möglich ist, Relationen verschiedener Arten zwischen zwei Räumen abzubilden. Die Ontologie baut auf der *Building Topology Ontology* auf und ist auch mit *ifcOWL* und den weiteren *LBD*-Ontologien kompatibel. Es werden *Competency Questions* definiert und anschließend mittels SPARQL-Abfragen und Beispielen gezeigt, dass die Ontologie diese erfüllt. Darüber hinaus, werden Möglichkeiten entwickelt, die entsprechenden nötigen Informationen aus IFC-Dateien in einen RDF-Graph zu überführen.

# Inhaltsverzeichnis

**Abkürzungsverzeichnis** V

**1 Einleitung** 1
- 1.1 Ausgangslage . . . 1
- 1.2 Modellbildung/ Annahmen . . . 2

**2 Stand der Wissenschaft** 4

**3 Grundlagen** 10
- 3.1 Building Information Modelling (BIM) . . . 10
- 3.2 Industry Foundation Classes (IFC) . . . 12
- 3.3 Linked Data und Semantic Web . . . 12
- 3.4 Resource Description Framework (RDF) . . . 14
- 3.5 Labeled Property Graph (LPG) . . . 15
- 3.6 Ontologien . . . 16
  - 3.6.1 RDFS und Web Ontology Language . . . 16
  - 3.6.2 IfcOWL . . . 18
  - 3.6.3 Building Topology Ontology (BOT) . . . 19
  - 3.6.4 Weitere Ontologien . . . 21
- 3.7 SPARQL . . . 21
- 3.8 Graphbasierte Regelprüfung . . . 24

**4 Ontologie** 25
- 4.1 Analyse der Fragestellungen . . . 25
- 4.2 Konzeption der Ontologie . . . 30
  - 4.2.1 Klassen . . . 30
  - 4.2.2 Object Properties . . . 31
  - 4.2.3 Datatype Properties . . . 32
  - 4.2.4 Zusammenfassung . . . 32

**5 SPARQL Queries** 35
- 5.1 Abfragen zu den Fragestellungen . . . 35
  - 5.1.1 Abfrage 1: Modellvollständigkeit . . . 35
  - 5.1.2 Abfrage 2: Erreichbarkeit . . . 36
  - 5.1.3 Abfrage 3: Mindestanforderungen . . . 37
  - 5.1.4 Abfrage 4: Raumbedürfnisse . . . 38
  - 5.1.5 Abfrage 5: Raumgruppen . . . 40
  - 5.1.6 Abfrage 6: Raumanzahl . . . 41
  - 5.1.7 Abfragen zur Optimierung . . . 43
- 5.2 Zusammenfassung . . . 44

**6 Umsetzung** **45**
6.1 Nachbarschaftsalgorithmus . . . 45
6.1.1 Überlegung zur Bestimmung horizontaler Nachbarschaften . . . 46
6.1.2 Überlegung zur Bestimmung vertikaler Überlagerungen . . . 51
6.1.3 Praktische Umsetzung der beiden Überlegungen . . . 53
6.2 Erreichbarkeit von Räumen . . . 58
6.3 Umwandlung von Property Sets . . . 58
6.4 Erstellung des Graphs . . . 58
6.5 Zusammenführung aller Prozesse . . . 59

**7 Anwendungsfälle** **60**
7.1 Beispiel 1: Bürogebäude . . . 60
7.1.1 Beschreibung des Gebäudes . . . 60
7.1.2 Fiktiver Regelsatz . . . 61
7.1.3 Auswertung des Regelsatzes . . . 62
7.1.4 Optimierung . . . 67
7.1.5 Zusammenfassung . . . 67
7.2 Beispiel 2: Betriebsräume eines U-Bahnhofs . . . 67
7.2.1 Ausgangslage . . . 67
7.2.2 Richtlinienkatalog . . . 68
7.2.3 Bahnhof Willibaldstraße . . . 69
7.2.4 Auswertung ausgewählter Regeln . . . 70
7.2.5 Zusammenfassung . . . 76

**8 Fazit** **77**

**A Pläne und Code-Beispiele** **79**
A.1 Grundrisse Bürogebäude . . . 79
A.2 Anforderungen Gleichrichterwerk . . . 82

**B Beigefügte Elemente** **84**

**Literatur** **86**

# Abkürzungsverzeichnis

| | |
|---|---|
| **2D** | zweidimensional |
| **3D** | dreidimensional |
| **AABB** | Axis Aligned Bounding Box |
| **AIA** | Auftraggeber-Informationsanforderungen |
| **BIM** | Building Information Modeling |
| **BOT** | Building Topology Ontology |
| **bsi** | BuildingSMART International |
| **CAD** | Computer Aided Design |
| **CDE** | Common Data Environment |
| **IFC** | Industry Foundation Classes |
| **IRI** | International Resource Identifier |
| **LBD** | Linked Building Data |
| **LPG** | Labeled Property Graph |
| **OBB** | Oriented Bounding Box |
| **OWL** | Web Ontology Language |
| **PDF** | Portable Document Format |
| **RAO** | Room Analysis Ontology |
| **RDF** | Resource Description Framework |
| **RDFS** | Resource Description Framework Schema |
| **SPARQL** | SPARQL Protocol and RDF Query Language |
| **STEP** | STandard for the Exchange of Product model data |
| **SWM** | Stadtwerke München |
| **TGA** | technische Gebäudeausrüstung |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |

# Kapitel 1

# Einleitung

## 1.1 Ausgangslage

Die Digitalisierung ist im Bauwesen zwar mittlerweile weit fortgeschritten, aber immer noch nicht überall angekommen und auch noch nicht für alle Fachbereiche fertig entwickelt. Bei vielen Akteuren aus den Bereichen Bau, Planung und Betrieb herrscht immer noch Uneinigkeit zwischen Personen, die auf den Einsatz von Building Information Modeling (BIM) und dessen begleitenden Technologien drängen, während andere noch nicht überzeugt sind, dass BIM für ihren speziellen Bereich bereits ausgereift ist. Dabei handelt es sich nur noch selten um Menschen, die sich grundsätzlich nicht auf eine neue Arbeitsweise einstellen wollen, denn mittlerweile kann ein Großteil der Firmen und Behörden, die BIM verwenden, auf erfolgreiche Pilotprojekte und gut laufende BIM-Projekte verweisen. Vielmehr ist es sinnvoll, BIM nicht nur des Namens wegen einzuführen, denn die Umstellung ist zunächst ein zeit- und kostenintensiver Prozess und man strebt danach seine Ressourcen im Sinne der Wirtschaftlichkeit nicht für etwas ausgeben, was keine Verbesserung bringt. Deshalb ist es auch weiterhin nötig, durch Forschung und Entwicklung, immer weiter neue Anwendungsfälle und Problemlösungen zu finden und auch zu zeigen.

In der U-Bahnplanung der Landeshauptstadt München wird noch konventionell mit 2D-Plänen und viel händischer Prüfung gearbeitet. Die Stadtwerke München (SWM) haben als Betreiber hingegen bereits in Sanierungs- und Umbauprojekten Erfahrungen mit BIM gesammelt und gemeinsam mit der Abteilung Ingenieurbau des Baureferats möchte man in naher Zukunft mit der U9-Nord das erste BIM-Neubauprojekt starten. Da die Planung aus Kapazitätsgründen weitestgehend an einen externen Planer vergeben werden muss, liegt der Fokus bezüglich der BIM-Umsetzung neben dem generellen BIM-Management auf der Bahnhofskonzeption, welche weiterhin intern durch die Architekten und Planer der technische Gebäudeausrüstung (TGA) umgesetzt wird. Dabei geht es unter anderem um die Lage und Vordimensionierung der Betriebsräume für die Grundlagenermittlung und die Vorplanung. Dadurch wird sichergestellt, dass ein Bahnhof bezüglich seiner Grundfläche bereits ausreichend dimensioniert ist und im weiteren Planungsverlauf somit möglichst keine aufwendigen Änderungen an den grundlegenden Maßen nötig sind. Dabei werden allerdings noch keine Kabeltrassen oder Einrichtungen explizit gezeichnet, sondern nur die Raumaufteilung festgelegt. Anschließend werden die resultierenden Grundrisspläne nach dem internen Richtlinienkatalog (RLK) geprüft. Der Richtlinienkatalog behandelt die Mindestanforderungen aller Betriebsräume hinsichtlich Geometrie, Materialien, technischer Ausstattung und wenn vorhanden, deren Beziehungen zu anderen Räumen oder deren Anforderungen im Gesamtkontext des Bauwerks oder der U-Bahnlinie. Eine reine

Prüfung dieser Anforderungen auf Basis von 2D-Plänen birgt deshalb Risiken bei der Koordinierung über mehrere Stockwerke hinweg. An diesem Punkt könnte schon der Umstieg auf ein 3D-Modell einen Vorteil bringen, aber mit der Verknüpfung der Informationen, die zu jedem Raum gehören und der anschließenden maschinellen Prüfung nach den Regeln des Katalogs lässt sich das volle Potenzial ausschöpfen.

Das Prinzip, Räume als Hauptinformationsträger zu nutzen und deren Relationen zu betrachten, ohne deren zukünftigen baulichen Inhalt und weitestgehend ohne das restliche Bauwerk um die Räume herum, ist Grundlage dieser Arbeit. Es soll untersucht werden, wie man Informationen und Verbindungen der Räume darstellen und auswerten kann, sodass sich der Richtlinienkatalog abbilden lässt. Dabei soll das Ergebnis auch allgemeingültig für ähnliche Probleme anwendbar sein. Es handelt sich damit abstrakt betrachtet um die Betrachtung und Auswertung einer Teilmenge eines BIM-Modells. Für die Analyse von komplexen verknüpften Daten und auch von vereinfachten Teilmodellen wird schon seit einigen Jahren die Anwendung von *Linked Data* für das Bauwesen erforscht. *Linked Data* ermöglicht es speziell Relationen zwischen Objekten darzustellen und Schlussfolgerungen auf Basis der Verknüpfung von Informationen zu generieren. Dieses Prinzip wird genutzt, wenn Informationen aus verschiedenen Quellen maschinenlesbar zusammengeführt werden müssen, um aus allen Quellen kombiniert neue Erkenntnisse zu gewinnen. Dabei geht es auch darum von spezialisierten Datenformaten weg zu einem neutralen, universell interpretierbaren, webbasierten Datencluster zu kommen, welches auch von zuvor unberücksichtigten Tools für neue Aufgaben ausgewertet werden kann, ohne neue Anpassungen an der Datenstruktur. In dieser Arbeit wird *Linked Data* dazu genutzt, um die Relationen zwischen Räumen darzustellen und zu analysieren. Dabei wird auf vorhandenen Teillösungen aufgebaut und neue Konzepte zur Lösung des Anwendungsfalls entwickelt.

Da die Konzepte, welche in dieser Arbeit verwendet werden, einige Abstraktionen und neue Begriffe beinhalten, werden diese im Folgenden vorgestellt.

## 1.2 Modellbildung/ Annahmen

Der gesamten Arbeit liegt folgendes Betrachtungskonzept zugrunde:
Vom gesamten BIM-Modell werden ausschließlich die Räume mit ihren topologischen Beziehungen und ihren Eigenschaften betrachtet. Auf dieses Konstrukt aus relativ zueinander platzierten Räumen werden die Eigenschaften und Bedürfnisse der Objekte übertragen, welche sich eigentlich in diesen Räumen befinden, oder diese verbinden. So geht die Notwendigkeit eines Wasseranschlusses für ein Waschbecken in einem Badezimmer über an das Badezimmer als Raum, welches nun stellvertretend diesen Anschluss fordert. Das Waschbecken muss dadurch noch nicht modelliert werden, denn der Raum an sich setzt bereits fest, dass er für seine Nutzung und Einbauten gewisse Versorgungsleitungen benötigt. Ebenso lässt sich das Bedürfnis der Menschen in einem Büro, eine Kaffeeküche oder einen Pausenraum in näherer Umgebung zu haben, auf ihren Büroraum übertragen,

von welchem aus ein Raum innerhalb einer vorgeschriebenen Distanz erreichbar sein soll, der die entsprechenden Anforderungen erfüllen kann.

Hierfür werden für diese Arbeit zwei Begriffe definiert:

Unter *Raumbeziehungen* werden alle Eigenschaften und Beziehungen der Räume untereinander zusammengefasst. Dazu zählt die Lage zweier Räume zueinander (angrenzend, darüber, darunter, im gleichen Stockwerk, etc.). Die Verbindungen zwischen einem Raum, der die Erfüllung eines Bedürfnisses anbietet und den Räumen die dieses abrufen zählen ebenfalls dazu. Wenn also z.B. aus einem Raum heraus ein Kabel, eine Leitung, oder ein gehbarer Weg zu einem anderen Raum möglich ist oder existiert, dann besteht eine Beziehung zwischen den Räumen.

Anders ist das bei den *Raumbedürfnissen*. Diese beschreiben die grundsätzlich vorhandenen Anforderungen eines Raumes unabhängig vom Erfüllungsgrad dieser. Bedürfnisse können, wie oben genannt, die Versorgung mit gebäudetechnischen Einrichtungen wie Strom, Wasser, Heizung, etc. sein. Es können aber auch Anforderungen aus der Nutzung sein. Raumbedürfnisse können ebenfalls sein, dass ein Raum neben einem anderen Raum liegen muss, oder das Gegenteil davon. Ebenso, ob gewisse Räume in der Nähe sind, oder als Gruppe zusammenhängen. Zudem lassen sich Angebot und Nachfrage-Beziehungen abbilden, ähnlich der Gebäudetechnik. Als Beispiel kann ein Laborraum mit hochpräziser Messtechnik die folgenden Bedürfnisse haben:

- Strom, Ab- und Frischwasser
- Heizung und Kühlung
- Belüftung mit Filterung
- Sprinklerung
- nur über einen Raum betretbar, der die Funktion einer Luftschleuse erfüllt
- darf nicht neben Räumen liegen, die Erschütterungen erzeugen (z.B. durch Maschinen)
- Zugang nur für ausgebildetes Personal

Bedürfnisse, welche im Raum selber erfüllt werden können, zählen zunächst nicht dazu, werden aber im Rahmen dieser Arbeit mitberücksichtigt. Wenn z.B. in einem Raum ein Erste-Hilfe-Kasten nötig ist, dann ist dies keine Anforderung, welche durch einen anderen Raum erfüllt werden kann. Aber da solche Regeln, wie z.B. Anforderungen an die Größe der Grundfläche, die Höhe, oder an die Anzahl an Steckdosen oder Lampen häufig abhängig von Raumbedürfnissen sind, sollten sie mitbetrachtet werden.

# Kapitel 2

# Stand der Wissenschaft

Damit das Ergebnis dieser Arbeit nicht nur für eine ganz bestimmte Softwarekombination funktioniert, soll nach dem Prinzip des Big-Open-BIM (BORRMANN et al., 2021) ein möglichst neutrales, standardisiertes und weit verbreitetes Datenformat als Datengrundlage genutzt werden. Ein solches Format sind die *Industry Foundation Classes (IFC)* (BUILDINGSMART INTERNATIONAL, 2019). Dieses ISO-zertifizierte Format wird bereits seit Jahren in der gesamten Baubranche für den Austausch von BIM-Modellen genutzt. Genauere Erläuterungen dazu folgen im nächsten Kapitel.

Aufgrund der Relevanz von *IFC* gab es in den letzten Jahren bereits viele Untersuchungen zu Datenabfragen und der Zugänglichkeit von einzelnen Daten in diesem Format und dem übergeordneten *EXPRESS*-Schema. Unter den Experten auf diesem Gebiet herrscht große Einigkeit, dass *IFC* in seiner ursprünglichen Form als *STEP*-File für gezielte Abfrage von einzelnen oder verknüpften Daten ungeeignet ist. Die wiederkehrenden Kritikpunkte zum Beispiel bei ZHU et al. (2023), PAUWELS und TERKAJ (2016) und TAUSCHER et al. (2016) sind, dass die zugrundeliegende Struktur des *EXPRESS*-Schemas für solche Zwecke nicht ausgelegt ist und dass für die vorhandenen Möglichkeiten jeweils sehr gute Kenntnisse über *IFC* nötig sind, was den Anwenderkreis stark beschränkt. Des weiteren bemängeln sie, dass *IFC* außerhalb der Bauwirtschaft nicht bekannt ist und daher die verfügbare Softwarelandschaft eher klein ist und die Verknüpfungsmöglichkeiten zu anderen Fachbereichen gering sind. BEETZ et al. (2009) und TECLAW et al. (2023) merken an, dass *IFC* zwar als ISO-Standard verlässlich und akzeptiert ist, aber z.B. für Nischenbereiche und Forschung unflexibel und nur schwer erweiterbar ist.

Zusätzliche *IFC*-Formate *IfcXML* und *IfcJSON* lösen zwar das Problem der Interoperabilität mit anderen Disziplinen, verbessern aber nicht direkt die Abfragemöglichkeiten von Daten (ZHU et al., 2023). Es gibt Versuche spezielle, auf *IFC* abgestimmte BIM-Abfragesprachen zu entwickeln. Dazu zählen z.B. *BIMQL* (MAZAIRAC & BEETZ, 2013), *QL4BIM* und *VCCCL* (PREIDEL et al., 2017). Diese Sprachen sind aber teilweise nur auf bestimmte Fachbereiche abgestimmt und selbst wenn vereinfachende Befehle implementiert wurden, so benötigt man doch meist fundierte Kenntnisse über das *IFC*-Schema, oder Programmieren an sich (TAUSCHER et al., 2016 und ZHU et al., 2023).

Als Lösung werden graphbasierte Datenformate vorgeschlagen, die in der Lage sind alle, oder einen Teil der Informationen und Beziehungen eines *IFC*-Modells nicht nur zu speichern, sondern auch auf effizientere Weise für Abfragen und Folgerungen zur Verfügung zu stellen. Während TAUSCHER et al. (2016) und ZHU et al. (2023) ihre Konzepte auf gerichteten Graphen bzw. *Property Graphs* aufbauen, nutzen BEETZ et al. (2009), BARBAU et al. (2012) und PAUWELS und TERKAJ (2016) das *Resource Description Framework*

*(RDF)* und das dazugehörende Prinzip der Ontologien. Sie sehen den Vorteil darin, dass *RDF* bzw. das darüber stehende Prinzip des *Semantic Web* (BERNERS-LEE et al., 2001) flexibel ist, man seine Bauwerksinformationen leicht mit fremden Datensätzen verknüpfen kann, seine eigenen Regelsätze und Ontologien für seinen Graph definieren kann und dass dieses Format sowohl für Menschen, als auch Maschinen nicht nur lesbar sondern auch interpretierbar ist. Aus diesen Ontologien ging zunächst *ifcOWL* (BEETZ et al., 2009) hervor, welche das *IFC*-Schema vollständig und ohne große Veränderungen abbildet. Damit erbt *ifcOWL* aber die Komplexität des ursprünglichen Schemas und ist nicht auf das *RDF*-Schema optimiert, was zudem noch zu gestiegenen Dateigrößen gegenüber des originalen Formats führt. Angetrieben von diesen Problemen gab es zwei Ansätze. Mit *SimpleBIM* versuchte PAUWELS und ROXIN (2016) *ifcOWL* so zu vereinfachen und anzupassen, dass die Graphen kleiner und effizienter werden, ohne dabei Informationen zu verlieren. Die Ontologie erzielt dabei eine Einsparung von ca. 90 % bezüglich Datei- und Graphgröße gegenüber *ifcOWL*. In beiden Ontologien wird die Geometrie nicht berücksichtigt, da sie in Graphen nur schlecht dargestellt werden kann. Eine konsequente Abbildung von Geometrie als Graph würde die Datenmenge um ein vielfaches vergrößern und damit den Rechenaufwand für jede Abfrage deutlich steigern, da jedes Element soweit zerlegt wird, dass sogar ein Punkt noch in seine drei Koordinatenwerte aufgeteilt wird. Da es oft nicht nötig ist, tatsächlich alle Informationen vollständig zu erhalten, verfolgt die *Linked Building Data Community Group* 2024 den Ansatz mehrere Ontologien für bestimmte Zwecke und Gebiete zu entwickeln oder bestehenden Ontologien von anderen Fachbereichen zu nutzen und diese dann nach Bedarf zu verknüpfen, damit die gespeicherte und zu verarbeitende Informationsmenge möglichst klein bleibt. Mit dem *IFCtoLBD*-Converter (BONDUEL et al., 2018) lassen sich *IFC*-Dateien modular in einzelne Graphen entsprechend der ausgewählten Ontologien von Linked Building Data (LBD) umwandeln. Zu den implementierten Ontologien zählt unter anderem die *Building Topology Ontology (BOT)* (RASMUSSEN et al., 2021). Diese lässt die Beziehungen zwischen Zonen (Räume, Geschosse, etc.) und Elementen (Wände, Türen, Decken, etc.) darstellen und bildet damit das Grundgerüst für alle anderen Ontologien, denn sie repräsentiert im wesentlichen die topologische Gebäudestruktur. Das Gerüst basiert weitestgehend auf Aussagen bezüglich der Lage von Zonen und Elementen. Eine Zone kann neben einer anderen Zone liegen, innerhalb dieser, oder beide Zonen können sich überschneiden. Die gleichen Aussagen können auch für Elemente untereinander und zwischen Elementen und Zonen getroffen werden.

Diese Ontologie dient als Inspiration und Grundlage für diese Arbeit, da sich hier bereits die topologischen Beziehungen von Räumen untereinander auf eine gewisse Weise darstellen lassen können. Da aber Räume und Raumeigenschaften als getrennte Ontologien vorliegen und auch das Raumgefüge nicht eindeutig abbildbar ist, wird eine auf *BOT* basierende Erweiterung vorgeschlagen. Diese Ontologie soll Raumbeziehungen und Raumbedürfnisse kompakt abbilden können, ohne weitere aufwendige Berechnungen zu erfordern. Gleichzeitig soll diese Ontologie möglichst mit anderen Ontologien, speziell denen von LBD kompatibel sein. Ein Graph für horizontale Raumbeziehungen wurde auch bereits in LANGENHAN et al. (2013) erstellt (siehe untere Grafik), jedoch ohne vertikale

Komponente, in einem anderen Graph-System und mit vereinfachenden Annahmen bei der Umsetzung. Die Vereinfachungen beruhen zum Beispiel darauf, dass an eine Wand höchstens zwei Räume angrenzen dürfen (auf beiden Seiten je einer). Dies ist aber nicht immer der Fall, weswegen in Kapitel 6 ein neuer Lösungsansatz entwickelt wird. Ebenso wird ein Algorithmus entworfen, der die vertikalen Raumbeziehungen erfassen kann.

Mit der geometrischen Berechnung der topologischen Lage von Räumen zueinander hat sich zum Beispiel DAUM und BORRMANN (2014) bereits befasst, allerdings wird dort nur berechnet, ob sich zwei Geometrien schneiden, berühren, ineinander liegen, oder nichts miteinander zu tun haben. Da benachbarte Räume aber meistens durch Wände getrennt sind und sich daher nicht physisch berühren, muss für diese Arbeit ein neues Berechnungskonzept dafür geschaffen werden.

Zusammenfassend lässt sich also sagen, dass *Linked Data* im Bauwesen bereits genutzt wird und von einer breiten Masse an Forschenden weiterhin untersucht und weiterentwickelt wird. Man hat sich mit *ifcOWL* und den Ontologien von LBD eine gemeinsame Grundlage geschaffen, auf welcher auch in dieser Arbeit aufgebaut werden kann. Die Relationen zwischen Räumen wurden bereits erfolgreich als Graph modelliert und genutzt, können aber die Problemstellung dieser Arbeit noch nicht vollständig und effizient lösen.

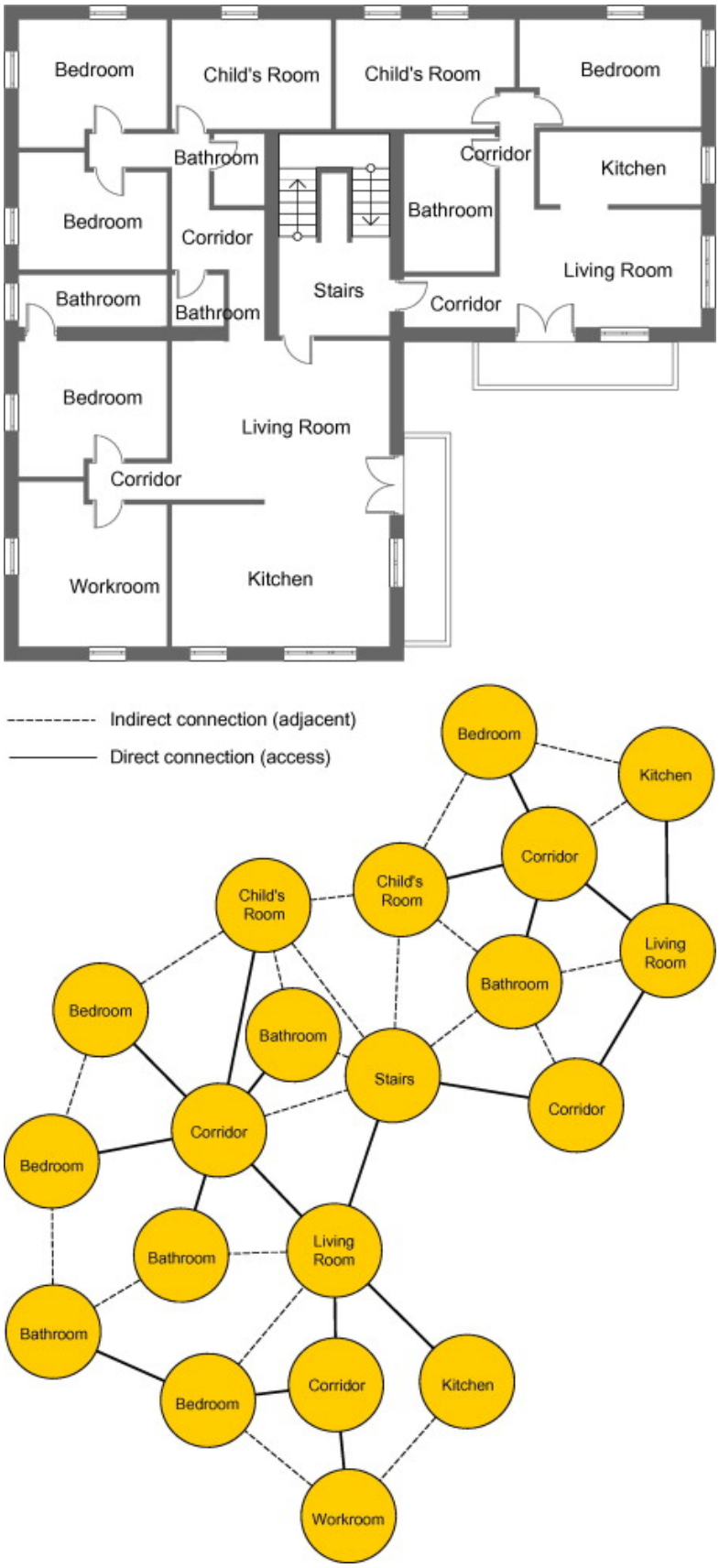



Abbildung 2.1: Darstellung von Raumbeziehungen in LANGENHAN et al. (2013), ähnlich dem in dieser Arbeit angestrebten Konzept.

Während also die topologische Darstellung der Räume und auch prinzipiell die Zuordnung von Informationen keine ungelösten Angelegenheiten mehr sind, so fehlt es noch an den konkreten Analysemethoden und einer speziell auf das in der Einleitung beschriebene Problem zugeschnittene Lösung. Um die grobe Problembeschreibung aus der Einleitung in diesem Kontext bearbeiten und einordnen zu können, werden im Folgenden sieben konkrete Fragestellungen formuliert, welche durch die in dieser Arbeit entwickelte Ontologie beantwortet werden können sollen. Die Fragen dienen dabei als *Competency Questions (QC)* für die Ontologie, also als Fragen welche die zu entwickelnde Ontologie mindestens korrekt beantworten können muss (ALLEMANG, 2011). Gleichzeitig bilden sie die Struktur dieser Arbeit. Anhand dieser Fragen wird die Ontologie in den folgenden Kapiteln entwickelt und getestet.

Die zu beantwortenden Fragen lauten:

- **Sind alle geforderten Räume modelliert?** Bei dieser Frage geht es um die grundlegende Modellvollständigkeit. Es sollte geprüft werden können, ob alle Räume bzw. Raumtypen, die nach einer Richtlinie, oder grundsätzlich einer Liste im Modell vorhanden sein sollen auch modelliert wurden. Dabei handelt es sich nicht um ein neues und unerforschtes Kriterium, da so eine Abfrage über Prüf-Tools wie *Solibri* (SOLIBRI, 2024) bereits lange möglich ist und auch andere Ontologien bereits in der Lage sind Raumnamen oder -typen darzustellen. Aber eine Ontologie, die Raumbeziehungen und -bedürfnisse abbilden kann, sollte auch dazu in der Lage sein.

- **Sind alle Räume zugänglich?** Ob man einen Raum betreten kann, hängt davon ab, ob er mindestens eine Tür hat. Diese Verbindung *Raum - Tür - Raum* ist ein beliebtes Beispiel, um die Funktionsweise von *IFC* zu erklären und es ist daher bereits bekannt und wurde auch bereits z.B. von LANGENHAN et al. (2013) und ZHU et al. (2023) zur Verknüpfung von Räumen in einem Graph genutzt. Da die Zugänglichkeit zu einem Raum eine Beziehung zwischen zwei Räumen herstellt und auch ein Bedürfnis sein kann (z.B. Raum *X* darf nur über Raum *Y* betreten werden), sollte es in dieser Arbeit mitberücksichtigt werden und eine Ontologie diese Beziehung explizit darstellen können, um die mehrschrittige Abfrage aus den *IFC*-Daten zu vereinfachen.

- **Werden Mindestanforderungen eingehalten?** Dabei geht es um die Prüfung der Eigenschaften der Räume gegen ein beliebiges Regelwerk. Es soll geprüft werden können, ob ein Raum von einem bestimmten Typ z.B. über die nötige Ausstattung verfügt (wenn diese als Information mit diesem Raum verknüpft ist) oder den Mindestanforderungen an seine Geometrie genügt (Grundfläche, Volumen, Raumhöhe, etc.). Es soll eine Lösung gefunden werden, wie die Raumbedürfnisse bei der Modellierung berücksichtigt werden können, sodass diese später als Graph effizient ausgewertet werden können.

- **Werden alle Bedürfnisse erfüllt?** Es soll prüfbar sein, ob für jedes angeforderte Bedürfnis auch eine Quelle vorhanden und erreichbar ist. Dafür gilt es einen Weg zu

finden, Bedürfnisse bzw. die dafür nötigen Ressourcen sowohl als Angebot als auch als Nachfrage eines Raums zu definieren und diese möglichst allgemeingültig für alle möglichen Arten von Bedürfnissen.

- **Liegen Raumgruppen zusammen?** Diverse Regelwerke, stellen nicht nur Anforderungen an die einzelnen Räume, sondern auch an die Lage dieser Räume zueinander. Ein Beispiel dafür wäre, dass alle Räume, die zu einer Wohneinheit gehören als eine Gruppe aneinander liegen sollen und nicht durch andere Räume getrennt werden dürfen. Eine Erkennung solcher Zusammenhänge ist mit den vorhandenen Ontologien nicht direkt möglich.

- **Sind genügend Räume vorhanden?** Es gibt Räume, die müssen in einem Gebäude in gewissen Abständen, je Etage, pro einer gewissen Anzahl an anderen Räumen oder nach gewissen Kriterien mehrfach vorhanden sein. Auch solche Überprüfungen sollten leicht abzufragen sein.

- **Liegen thematische Räume günstig zueinander?** Neben der reinen Konformität gegenüber einem Regelsatz kann ein Gebäude mehr oder weniger effizient gestaltet sein. Es sollten Aussagen darüber getroffen werden können, ob eine Grundrissvariante besser hinsichtlich eines Kriteriums ist, als eine andere. Da in der Regel sehr viele Bedürfnisse und Anforderungen gleichzeitig erfüllt werden müssen, ist eine ganzheitliche Optimierung ein komplexes, teils subjektives und projektspezifisches Unterfangen.

# Kapitel 3

# Grundlagen

Das folgende Kapitel soll einen Überblick über alle zugrunde gelegten Konzepte geben.

## 3.1 Building Information Modelling (BIM)

Building Information Modeling (BIM) steht für die Digitalisierung und Vereinigung des Bauwesens, von der Planung, über die Ausführung, bis zum Betrieb. Es steht für eine modellbasierte, kollaborative, transparente und standardisierte Arbeitsweise über die Grenzen des eigenen Fachbereichs hinaus, mit Fokus auf einer kontinuierlichen und widerspruchsfreien Datenstruktur (VDI 2552 BLATT 1: 2020-07).

> *„BIM basiert auf der konsequenten Nutzung und Wiederverwendung digitaler Daten und hilft, die Produktivität zu erhöhen und gleichzeitig die Fehlerquote zu senken, da Fehler erkannt und behoben werden können, bevor sie zu ernsthaften Problemen werden.“* BORRMANN et al., 2021

BIM ist aber kein klar vorgeschriebener Prozess. Jede Firma oder jedes Projektteam kann entscheiden, wie ihr BIM-Projekt ablaufen soll, welche Herangehensweise wird gewählt, welche Projektaspekte werden überhaupt in BIM umgesetzt und welche Ziele möchte man damit erreichen. Und auch zum gleichen Ziel führen mittlerweile mehrere Wege. Diese Aussage scheint zwar etwas konträr zum Ziel einer standardisierten Arbeitsweise, aber das ist zum einen der immer noch laufenden Weiterentwicklung geschuldet und es wird auch stetig an der Standardisierung auf mehreren Ebenen gearbeitet. Auf nationaler und internationaler Ebene wird z.B. in der DIN EN ISO 19650-1: 2019-08 definiert was genau als BIM bezeichnet wird, was mit den zugehörigen Begriffen gemeint ist und welche Prozesse zu BIM gehören. Daraus ergab sich die Bezeichnung „BIM nach ISO 19650“ als gängige Definition und als gemeinsame Basis. Eine Einordnung in den Gesamtkontex findet sowohl in der Norm selbst als auch in der VDI-Richtlinie 2552 Blatt 1 2020-07 statt, welche auf der ISO-Norm aufbaut. Dort werden vier Leistungsniveaus beschrieben, welche den BIM-Reifegrad eines Projekts oder einer Arbeitsweise beschreiben.

*Level 0* beschreibt dabei die traditionelle Zeichenarbeit auf Papier oder isolierten CAD-Systemen mit Datenaustausch über Formate, die nur durch Menschen interpretiert werden können, z.B. in E-Mails, PDFs oder über gedruckte Pläne. Eine Automatisierung ist dort nur lokal begrenzt möglich und Standardisierungen, wie z.B. die DIN 1356-1: 2024-04 für einheitliche Plansymbole, beziehen sich mehr auf die einheitliche Lesbarkeit und eindeutige menschliche Interpretation, als auf den digitalen Datenaustausch.

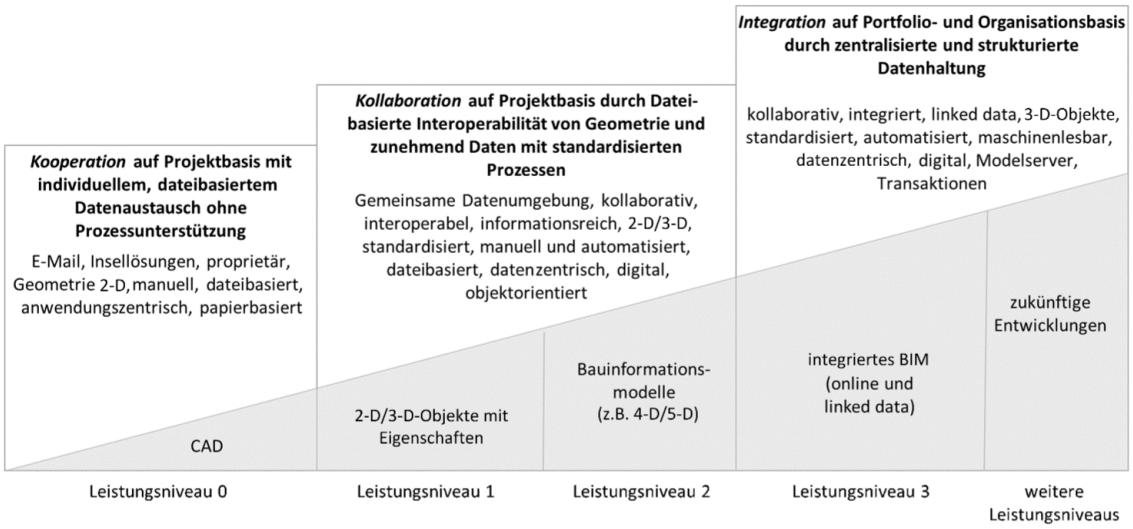



Abbildung 3.1: Übersicht der Leistungsniveaus VDI 2552 BLATT 1: 2020-07

*Level 1* bedingt bereits einen gemeinsame Datenplattform (CDE) und die Arbeit mit 3D-Geometrie. 2D-Zeichnungen sollen nur noch unterstützend verwendet werden und eine automatisierte Auswertung von Daten sollte bereits möglich sein. Es sind erste Informationsverknüpfungen vorhanden.

*Level 2* ist, was vorhin als „BIM nach ISO 19650“ beschrieben wurde. Es werden 3D-Modelle mit verknüpften Informationen genutzt. Es wird dabei datenorientiert gearbeitet, das heißt die Interpretierbarkeit der Daten für Software ist im Prozessablauf wichtiger als die Zugänglichkeit zu diesen für Menschen. Dies ermöglicht weitreichende automatische Analysen und Simulationen und Wertschöpfung neuer Informationen durch systematische Kombination von vorhandenen Daten. Der Austausch auf CDEs erfolgt jedoch immer noch über, im besten Fall neutrale, Dateiformate. Das bedeutet aber meistens, dass Informationsverknüpfungen über Dateigrenzen hinweg erst durch spezialisierte Programme möglich werden.

*Level 3* ist dann der letzte konsequente Schritt in dieser Reihe. Einzelne Datengruppen in separierten Dateien gibt es nicht mehr. Auf der CDE gibt es nur noch einen unstrukturierten Datenhaufen, welcher rein maschinell interpretierbar ist und nicht mehr an explizite Software oder Softwarekombinationen gebunden ist. Alles ist auf die digitale Verwertung der Daten optimiert und es gibt über ein Projektzeitraum hinweg keine Datenverluste durch Umwandlung oder Weitergabe mehr. Die Informationen über z.B. Geometrie, Bauablauf, Materialien oder Baufortschritt sind in einer Datenbank so strukturiert, dass sie von beliebigen Programmen über standardisierte Schnittstellen systematisch und auf den Anwendungsfall und das Programm zugeschnitten extrahiert werden können. Das Prinzip *Linked Data*, auf dem diese Arbeit basiert lässt sich diesem Level zuordnen. Der Einsatz von Graphdatenbanken allein, rechtfertigt allerdings noch nicht die Einordnung des gesamten BIM-Prozesses auf Stufe 3 und zudem werden weitere, präziser definierte Stufen diskutiert. Denn auch mit *Linked Data* arbeitet man noch auf eine Weise dateibasiert, was je nach Definition im Widerspruch zum Leistungsniveau 3 steht (RASMUSSEN et al., 2020). Eine Erklärung zu *Linked Data* folgt in Kapitel 3.3.

Ein standardisiertes Austauschformat für BIM-Modelle, also deren Geometrie und damit verknüpften Informationen ist *IFC (Industry Foundation Classes)* von BuildingSMART International (bsi) (2019). Dieses Format ist in mehreren Versionen ISO-Zertifiziert (aktuell ISO 16739-1:2024) und wird stetig weiterentwickelt, um immer mehr Anwendungsfälle und Spezifikationen abzudecken. Die letzten Entwicklungen galten speziell dem Ingenieur- und Tiefbau. (BUILDINGSMART INTERNATIONAL, 2024)

## 3.2 Industry Foundation Classes (IFC)

*IFC* wird seit 1997 im *EXPRESS*-Schema entwickelt, welches objektorientierte Datenstrukturen definieren kann (ISO 10303-11, 2004-11). Das heißt es gibt Objekte, Attribute und Beziehungen. (BORRMANN et al., 2021)

Ein Beispiel, wie Zusammenhänge in *IFC* repräsentiert werden, ist in der folgenden Abbildung zu sehen. Darin wird die Verbindung zweier Räume über ihre gemeinsame Trennwand dargestellt. Die beiden Räume (*IfcSpace*) werden begrenzt (*BoundedBy*) durch ein Raumumgrenzungsobjekt (*IfcRelSpaceBoundary*). Diese Beziehungsobjekte haben jeweils die Wand (*IfcWallStandardCase*) als Bezugselement (*RelatedBuildingElement*). Umgekehrt liefert die Wand Begrenzungsflächen für zwei Beziehunginstanzen (*Provides-Boundary*) und diese gehören wiederum zu je einem Raum (*RelatingSpace*). Eine direkte Beziehung zwischen den beiden Räumen besteht allerdings nicht.

Aber mittlerweile gibt es *IFC* auch in weiteren Datenformaten außer *EXPRESS* bzw. *STEP* (Dateiformat zum *EXPRESS*-Schema) (BUILDINGSMART INTERNATIONAL, 2024). Dazu zählt auch *ifcOWL*, was in einem eigenen Kapitel (3.6.2) erklärt wird. Wie bereits in Kapitel 2 angesprochen ist *IFC* nicht gut für Abfragen von einzelnen Datenmengen geeignet und auch nur schwer mit anderen (*STEP*-)Dateien verknüpfbar (BEETZ et al., 2009). Es ist hierbei allerdings anzumerken, dass *IFC* auch nicht für die kritisierten Anwendungsfälle entwickelt wurde, weswegen es sich anbietet, dafür parallel eine Lösung anzubieten, ohne das eigentliche Format anzutasten (PAUWELS & TERKAJ, 2016). Die Wahl fiel bei vielen Forschungsprojekten auf das Prinzip *Linked Data* und Graphen.

## 3.3 Linked Data und Semantic Web

*Linked Data* und das zugrundeliegende *Semantic Web* sind keine Erfindung aus dem Bauwesen, sondern eng verbunden mit der Entwicklung des Internets. Nachdem Daten im Internet meistens dokumentbasiert abgelegt und übertragen wurden und die beteiligten Maschinen nur Transporter waren, entschied sich dessen Erfinder Tim Berners-Lee dazu, es um einen Bereich zu erweitern, in dem die gespeicherten Daten nicht nur maschinenlesbar, sondern auch interpretierbar sind: das *Semantic Web*. (BERNERS-LEE et al., 2001) Dabei geht es darum, dass Daten so abgebildet werden können, dass nicht nur der Mensch, sondern auch ein Computer Schlüsse daraus ziehen und Informationen aus mehreren Quellen verknüpfen kann. Zum Beispiel könnte ein Computer in einem umfassenden

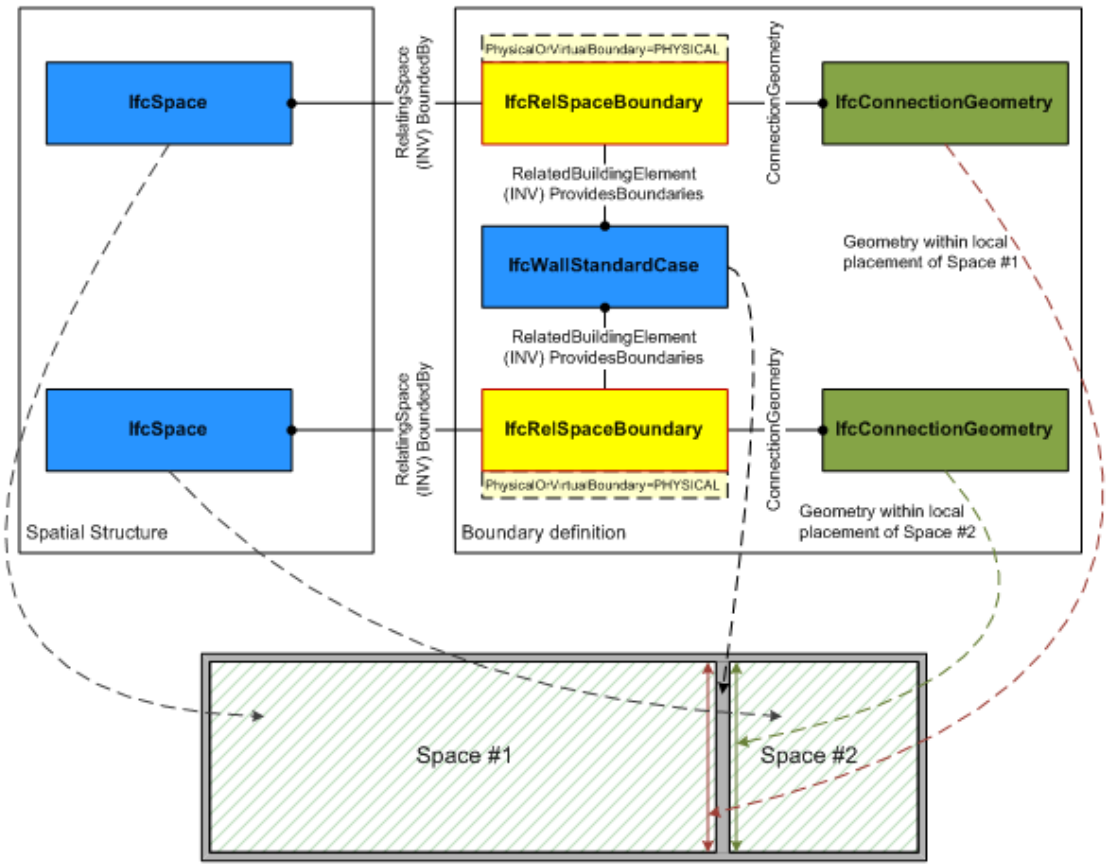



Abbildung 3.2: Beziehungen zweier Räume zueinander über eine gemeinsame Wand im IFC-Schema (BUILDINGSMART INTERNATIONAL, 2023)

*Semantic Web* bei der Suche nach einem Urlaubsort sofort die Flugzeit und den Preis dafür, sowie die prognostizierten Wetterdaten vor Ort, stattfindende Veranstaltungen, freie Hotels und wissenswertes zum Zielort finden, weil alle Informationsanbieter ihre Daten so zur Verfügung stellen, dass sie beliebig kombinierbar sind. Gleichzeitig kann über die Semantik sichergestellt werden, dass wenn eine Person den „Big Ben in London“ sehen will, keine Informationen über das London im Bundesstaat Kentucky, USA verwendet werden, denn wer sich geografisch nicht so gut auskennt, der würde vielleicht bei der Aussage „London liegt westlich von Manchester“ noch nicht vermuten, dass man auf dem falschen Kontinent unterwegs ist. Das Verknüpfen von Daten zu einem gemeinsamen Cluster für übergreifende Informationsgewinnung auf Basis des *Semantic Web* wird *Linked Data* genannt (BIZER et al., 2009). Damit Informationen sich nicht überlagern und stets zuordenbar sind, sollen sie nach BERNERS-LEE (2006) immer einen *Uniform Resource Identifier (URI)* besitzen, also eine eindeutige Adresse (meist eine Web-Adresse (URL), oder z.B. auch ISBNs für Bücher). In dieser Arbeit sind die für selbst erzeugte Daten verwendeten *URLs* fiktiv und nicht über das Internet auffindbar. Eindeutige Informationen allein reichen aber noch nicht für die reibungslose Verknüpfung beliebiger Daten von unabhängigen Bereitstellern. Um diese Informationen bzw. Ressourcen einheitlich verknüpfen zu können wurde das *Resource Description Framework (RDF)* entwickelt.

## 3.4 Resource Description Framework (RDF)

Das *Resource Description Framework (RDF)* ist unabhängig von Dateiformaten und stellt lediglich die Regeln für die Datenstruktur auf. Alle Informationen in *Resource Description Framework (RDF)* werden als Tripel gespeichert. Jedes Tripel besteht wie ein einfacher Satz aus *Subjekt*, *Prädikat* und *Objekt*. Dabei muss es sich nicht zwingend um exakt drei Wörter handeln, sondern es geht darum, dass einem Informationsobjekt (*Subjekt*) über eine Verbindung (*Prädikat*) entweder ein anderes Informationsobjekt (*Objekt*) oder ein expliziter Wert zugewiesen wird. (W3C, 2014a) Das folgende Beispiel soll dieses Prinzip veranschaulichen:

Algorithmus 3.1: Beispielhafte RDF-Aussagen in Pseudocode

```
1 Subjekt     Prädikat        Objekt
2
3 Efeu        ist eine        Pflanze.
4 Pflanzen    haben die Farbe grün.
5
6 Sandra      ist eine        Person.
7 Sandra      mag die Farbe   grün.
```

Ein Mensch kann diese Aussagen eindeutig lesen und daraus schließen, Sandra könnte Efeu, oder zumindest Pflanzen mögen. Aber wir Menschen wissen, was eine Farbe ist und dass *Pflanze* für die grammatikalische Korrektheit in der zweiten Zeile im Plural vorkommt, aber trotzdem das gleiche Objekt ist. Ein Computer kennt aber weder das Konzept von Farben noch Pluralformen. Er interpretiert nur, dass ein *Subjekt* über ein *Prädikat* mit einem *Objekt* verbunden ist. Ein Schritt in diese Richtung sind die oben angesprochenen *URIs* bzw. *IRIs* (*International Resource Identifier*, *URIs* mit Unterstützung für nicht-*ASCII*-Zeichen)(W3C, 2014a). Dadurch bekommt jede Ressource ihre eindeutige Adresse und es zeigt sich, ob zwei Objekte mit dem gleichen Namen auch tatsächlich exakt das gleiche sind. Den Ressourcen werden nun fiktive *URIs* zugeordnet:

Algorithmus 3.2: Beispielhafte RDF-Aussagen in Pseudocode mit URIs

```
1 <Subjekt> <Prädikat> <Objekt>
2
3 <http://www.natur.de/Efeu> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
     type> <http://www.natur.de/Pflanze>.
4 <http://www.natur.de/Pflanze> <http://www.farblehre.com/zuordnung/
     hat_die_Farbe> <http://www.farblehre.com/Farben/grün>.
5
6 <http://www.Daten.org/Sandra> <http://www.w3.org/1999/02/22-rdf-syntax-
     ns#type> <http://xmlns.com/foaf/0.1/Person>.
7 <http://www.Daten.org/Sandra> <http://www.farblehre.com/zuordnung/
     mag_die_Farbe> <http://www.farblehre.com/Farben/grün>.
```

Durch die *URIs* schwindet die Lesbarkeit für den Menschen drastisch, dafür ist jetzt zu erkennen, dass es sich beides mal um die gleiche Farbe *grün* und die gleiche Kategorie *Pflanze* handelt. Es sind auch *URIs* zu sehen, die nicht für dieses Beispiel erfunden

wurden. Für *RDF* gibt es auch bereits vorgefertigte Vokabulare zum deklarieren von Ressourcen. Für die Abbildung von Personen und Kontakten gibt es das *Friend of a Friend (FOAF)* Vokabular welches unter der angegebene Adresse (http://xmlns.com/foaf/0.1/) im Netz auffindbar ist. Zu RDF selbst gibt es auch ein Vokabular für die Definition von Klassen, Eigenschaften, Unterklassen und -eigenschaften und zum deklarieren von Instanzen davon (W3C, 2014a). Wir sagen also *Efeu* ist vom Typ *Pflanze* und *Sandra* vom Typ *Person* (unter der Annahme, dass die Klassen bereits definiert wurden). Um nicht jedes Mal den vollen Adresstext schreiben zu müssen, gibt es, wie in vielen Programmiersprachen, sogenannte *Namespaces*. Das heißt man definiert ein Präfix zu Beginn, das den allgemeinen Adressteil ersetzt:

Algorithmus 3.3: Beispielhafte RDF-Aussagen in Pseudocode mit Präfix

```
1  Präfix: nat:  <http://www.natur.de/>
2  Präfix: rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  Präfix: foaf: <http://xmlns.com/foaf/0.1/>
4  Präfix: farb: <http://www.farblehre.com/zuordnung/>
5  Präfix: :     <http://www.Daten.org/>
6
7  nat:Efeu    rdf:type            nat:Pflanze.
8  nat:Pflanze farb:hat_die_Farbe  <http://www.farblehre.com/Farben/grün>.
9
10 :Sandra     rdf:type            foaf:Person.
11 :Sandra     farb:mag_die_Farbe  <http://www.farblehre.com/Farben/grün>.
```

Da die Farbe und die Farbzuordnung leicht unterschiedliche Adressen haben, gilt das Präfix nur für die Zuordnung. Solche Mischformen sind möglich und es könnte auch weiterhin jede Ressource mit ihrer vollen Adresse ausgeschrieben werden. Auch leere Präfixe sind erlaubt. Sollte eine Ressource nicht explizit mit Adresse gespeichert werden, kann diese auch als *Blank Node* dargestellt werden und besitzt dann nur einen dokumentinternen Namen (W3C, 2014a).

Für *RDF* stehen mehrere Datenformate zur Verfügung. Neben der ursprünglichen Umsetzung in *XML* und *N-Triples* gibt es zum Beispiel noch das *Turtle*-Format, welches in dieser Arbeit verwendet wird (ALLEMANG, 2011). Die oben stehenden Pseudocode-Beispiele sind in der Reihenfolge zunächst *fiktiv*, dann an *N-Triples* und zuletzt an das *Turtle*-Format angelehnt.

Die Stärke von *RDF* liegt darin, dass jeder selbst ein Vokabular bzw. eine Ontologie erstellen kann, welche auf die eigenen Anforderungen zugeschnitten ist, oder beliebige fremde Ontologien verwenden und kombinieren kann, um seine eigene Datenwelt aufzubauen. Was *Ontologie* bedeutet und wie sie funktionieren wird in Kapitel 3.6 erläutert.

## 3.5 Labeled Property Graph (LPG)

*Labeled Property Graph (LPG)* ist neben *RDF* ein weiterer Graphentyp. *LPG* unterscheidet sich von *RDF* dadurch, dass Attribute nicht nur an die Knoten geknüpft werden, sondern

auch die Kanten Informationen tragen können. Das Format eignet sich gut für Graphdatenbanken und ist auch etwas besser für Abfragen optimiert. Allerdings ist es dafür weniger gut geeignet um Datensätze miteinander zu verknüpfen und ist zudem mehr für den menschlichen Nutzen ausgelegt, als auf die Maschineninterpretierbarkeit. (ZHU et al., 2023, HOWARD, 2017)

*LPG* wird in dieser Arbeit nicht verwendet, da sich gezielt für das Konzept der Ontologien und LBD als Grundlage entschieden wurde.

## 3.6 Ontologien

Der Begriff *Ontologie* wurde aus der Philosophie in die Informatik übertragen. *Ontologie* bedeutet die „Lehre des Sein“ bzw. „des Seienden“ und ist vereinfacht die Beschreibung und Definition von dem was existiert (HESSE, 2005). In der Informatik sind Ontologien also Beschreibungen und Regelwerke, die einen bestimmten Sachverhalt, eine bestimmte Realität beschreiben. Es wird definiert, was in dieser Umgebung an Prinzipien, Objekten, Typen und Eigenschaften existiert und wie diese in Beziehung zueinander stehen. Wichtig ist dabei, dass eine Ontologie widerspruchsfrei ist und auf Logik basiert (W3C, 2012). Aus der Beschreibungslogik erben Ontologien im *Semantic Web* das Prinzip von *A-Box* und *T-Box* (DE GIACOMO & LENZERINI, 1996). Dabei enthält der terminologische Teil (*T-Box*), also die Beschreibung von Konzepten und Regeln und die *A-Box* enthält alle tatsächlich deklarierten Instanzen und deren Eigenschaften. Ontologien können Inhalte für beide Seiten enthalten. Auch in einer Ontologie können bereits Individuen festgelegt werden, die in der beschriebenen Welt auf jeden Fall existieren. Um die logischen Zusammenhänge in einer Ontologie prüfen zu können gibt es als Werkzeuge sogenannte *Reasoner*. Diese können ähnlich wie ein Mensch auf Basis der vorhandenen Aussagen (Axiome) Rückschlüsse und Konsequenzen ziehen und gegebenenfalls feststellen, ob die als grundlegende Wahrheit angegebenen Axiome sich durch Logikfehler widersprechen (W3C, 2012).

Für das Semantic Web werden Ontologien in der Regel in der *Web Ontology Language (OWL 2)* geschrieben.

### 3.6.1 RDFS und Web Ontology Language

Die ursprüngliche „Regelsprache“ für *RDF* ist das *RDF-Schema (RDFS)*, welches selbst in *RDF* definiert ist (W3C, 2014b). Es beinhaltet die Möglichkeit Klassen, Subklassen, Eigenschaften und Subeigenschaften, zu definieren, sowie mit *rdfs:domain* und *rdfs:range* die Möglichkeit Subjekt und Objekt eines bestimmten Prädikats automatisch einer Klasse zuzuordnen. Das folgende Beispiel zeigt die Verwendung von *rdfs:range*. Alle *URIs* die auf das Prädikat *farb:hat_die_Farbe* folgen gelten automatisch als Individuen der Klasse *farb:Farbe*. Somit gilt *:grün* in diesem Beispiel automatisch als Individuum der Klasse

*farb:Farbe*. Das *RDFS*-Vokabular besitzt aus historischen Gründen zwei Namespaces: *rdf* und *rdfs*, von zwei unterschiedlichen *URLs* (W3C, 2014b).

Algorithmus 3.4: Beispielhafte Verwendung von rdfs:range

```
1  Präfix: rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  Präfix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  Präfix: farb: <http://www.farblehre.com/zuordnung/>
4  Präfix: :     <http://www.Daten.org/>
5
6  ----- T-Box -----
7  farb:Farbe          rdf:type    rdfs:Class.
8  farb:hat_die_Farbe  rdfs:range  farb:Farbe.
9
10 ----- A-Box -----
11
12 :Ball       farb:hat_die_Farbe  :grün.
```

Wesentlich umfangreicher bei den Beschreibungsmöglichkeiten ist die *Web Ontology Language (OWL 2)* (W3C, 2012). Sie erweitert das *RDF-Schema* so, dass man damit logikbasierte und verknüpfbare konsistente Ontologien schreiben kann. *OWL* lässt sich ebenfalls mit den bekannten *RDF*-Syntax schreiben und erhält in der Regel das Präfix *owl:*. Alle grundlegenden Bestandteile von Aussagen lassen sich in drei Kategorien einteilen. *Individuals* sind alle tatsächlich vorhandenen einzigartigen Objekte, *Classes* sind die Kategorien bzw. Klassen und alle Beziehungen werden *Properties* genannt (W3C, 2012). Dazu lassen sich auch alle Relationen kategorisieren. Statements zwischen zwei *Individuals* heißen *Object Properties*, Zuweisungen von Werten zu einem Objekt *Datatype Properties* und die Darstellungen von Beschreibungen für die Ontologie heißen *Annotation Properties* (W3C, 2012). Diese Prinzipien sind auch in diversen Tools zur Erstellung und Bearbeitung von Ontologien, wie *Protégé* implementiert (MUSEN, 2015). *Protégé* enthält zudem Reasoner zur Prüfung der Ontologie und wird in dieser Arbeit für die Erstellung der Ontologie verwendet.

Neben der Unterteilung und Verkettung von Klassen und Subklassen, gibt es in *OWL* auch die Möglichkeit Klassen (und Individuen) gleichzusetzen, um z.B. zwei Ontologien zu verknüpfen, die jeweils eine Klasse mit der exakt gleichen Bedeutung haben. Ebenso kann festgelegt werden, dass ein Objekt nicht gleichzeitig ein Element von zwei bestimmten Klassen sein kann. Ein Individuum der Klasse Tier sollte z.B. nicht gleichzeitig als Pflanze deklariert werden können. Zudem können Relationen mit Bedeutungen in die Gegenrichtung (als Graph gesehen) erweitert werden. Beispiele dafür sind, dass man die Eigenschaft „Ehepartner“ als *symmetrisch* deklarieren kann und somit ein Reasoner feststellen kann, dass wenn Anna Ehepartner von Bob ist, auch Bob automatisch Ehepartner von Anna ist. Es kann definiert werden, dass *istKindVon* die *inverse* Beziehung von *istElternteilVon* ist und dass diese Relationen nicht in beide Richtungen gleichzeitig auftreten dürfen. Man kann Klassen über logische Operatoren wie *und*, *oder* und *nicht* kombinieren (z.B.: Die Klasse *Mutter* ist die Schnittmenge aus *Frau* und *Elternteil*). Des weiteren lassen sich z.B. Kardinalitäten für Relationen sowie Minimal- und Maximalwerte für Daten festlegen. Aus den weiteren Möglichkeiten sei noch hervorgehoben, dass man einzelne Objekte auch mit

Kommentaren und Labels versehen kann, um die menschliche Lesbarkeit zu erleichtern. (W3C, 2012)

Die vollständige Implementierung von *OWL* als *OWL 2 Full* , basierend auf *RDFS*-Semantik, ist uneindeutig Daher gibt es als Subset von *OWL 2* noch *OWL 2 DL* basierend auf Beschreibungslogik-Semantik, welche wiederum eindeutig interpretierbar ist (W3C, 2012). Zusätzlich kann *OWL 2 DL* weiter in eins von drei Profilen (*OWL 2 EL, OWL 2 QL* und *OWL 2 RL*) reduziert werden. Diese vereinfachen jeweils die Implementierung und Nutzung für einen bestimmten Zweck. Eine ausführliche Auseinandersetzung mit den Profilen ist zum Beispiel bei KRÖTZSCH (2012) nachzulesen und übersteigt den Umfang dieser Arbeit.

### 3.6.2 IfcOWL

*IfcOWL* ist die direkte Umsetzung des *IFC*-Schemas als Ontologie und wurde entwickelt, da *IFC* an sich ein sehr isoliertes Format ist, welches nicht gut für Abfragen geeignet ist, wie schon in Kapitel 2 erläutert wurde (BEETZ et al., 2009). Eine Beschreibung und Herleitung der Ontologie ist bei BEETZ et al. (2009) und PAUWELS und TERKAJ (2016) zu finden. Die Ontologie ist offiziell bei BuildingSMART als Alternative zum Original im *EXPRESS*-Schema gelistet (BUILDINGSMART INTERNATIONAL, 2024).

Einer der wesentlichen Unterschiede zwischen *IFC* und *ifcOWL* ist dass das *IFC*-Schema auf den Prinzipien der Objektorientierung basiert, während *ifcOWL* auf Beschreibungslogik basiert und somit eher eine fehler- und widerspruchsfreie Darstellung sichergestellt werden kann (TERKAJ & ŠOJIĆ, 2015). Hinzu kommt dass *IFC* eine Abgeschlossenheit der Daten zugrundelegt (*closed world assumption (CWA)*) und somit alles was nicht direkt als *wahr* deklariert wird oder nicht explizit als *existent* beschrieben wird als *falsch* oder *nicht existent* angesehen wird (PAUWELS & TERKAJ, 2016). Dem gegenüber basiert aber *Linked Data* und damit auch *ifcOWL* auf der *open world assumption (OWA)*. Wenn etwas nicht in der eigenen Ontologie oder dem eigenen Datensatz beschrieben ist, dann heißt das nicht, dass es nicht existiert. Man geht davon aus, dass man gar nicht alles abgeschlossen wissen kann und daher keine absoluten Annahmen getroffen werden können. Was nicht explizit als *wahr* oder *falsch* deklariert wurde bleibt demnach ungelöst, da die Möglichkeit besteht, dass die Verlinkung mit einem anderen Datensatz und neuen Ontologien die passende Antwort hervorbringt.

Da Geometrie selten explizit in ihren einzelnen Komponenten (Flächen, Kanten, Punkte, Punktkoordinatenwerte) abgefragt wird und diese den Graph bei einer reinen *RDF*-Übersetzung um ein vielfaches aufbläht, gibt es Überlegungen, die Geometrie in anderen Formaten in den Graph zu integrieren, sodass dritte Programme die Geometrie verwerten können und der Graph dabei möglichst kompakt bleibt. Auseinandersetzungen mit Geometrie-Ontologien und der Integration von neutralen Geometrieformaten in Kombination mit *ifcOWL* sind zum Beispiel bei PAUWELS et al. (2017) und WAGNER et al. (2020) zu finden.

Ähnliche „generelle“ Umsetzungen des *IFC*-Schema sind z.B. *ifcWOD* (de FARIAS et al., 2015) und das zuvor bereits angesprochene *SimpleBIM* (PAUWELS & ROXIN, 2016). Neben den Versuchen das IFC-Schema als ganzes zu übersetzen gibt es auch das Konzept das Schema in einzelne thematische Bereiche aufzuteilen und um eine Kern-Ontologie herum so modular zu erweitern, dass man nicht unnötig viele Daten mit in den Graphen aufnimmt. Dazu zählen *BIMSO* (NIKNAM & KARSHENAS, 2017) und die Ontologien von Linked Building Data (LBD) mit *BOT* als Kern (RASMUSSEN et al., 2017).

### 3.6.3 Building Topology Ontology (BOT)

Die *BOT* ist die Basis der Ontologiereihe der von LBD und dient dem Zweck ein grundlegendes topologisches Gerüst eines Gebäudes abzubilden, welches anschließend für beliebige Anwendungsfälle erweitert werden kann (RASMUSSEN et al., 2020). Das allgemeine Präfix ist *bot:* und unter dem Namespace *https://w3id.org/bot#* zu finden (RASMUSSEN et al., 2021).

Den Kern der Ontologie bilden die drei Klassen *bot:Zone*, *bot:Element* und *bot:Interface*. Dabei sind alle physischen Objekte eines Gebäudemodells, egal ob Bauwerk, Technik oder Einrichtung Instanzen der Klasse *bot:Element*. Alle räumlichen Volumen bzw. Begrenzungen wie Räume, Geschosse, Gebäude und das Grundstück sind als *bot:Zone* bzw. als jeweils eine Unterklasse davon definiert. Die Klasse *bot:Interface* repräsentiert alle Berührungsflächen zwischen Zonen und Elementen.

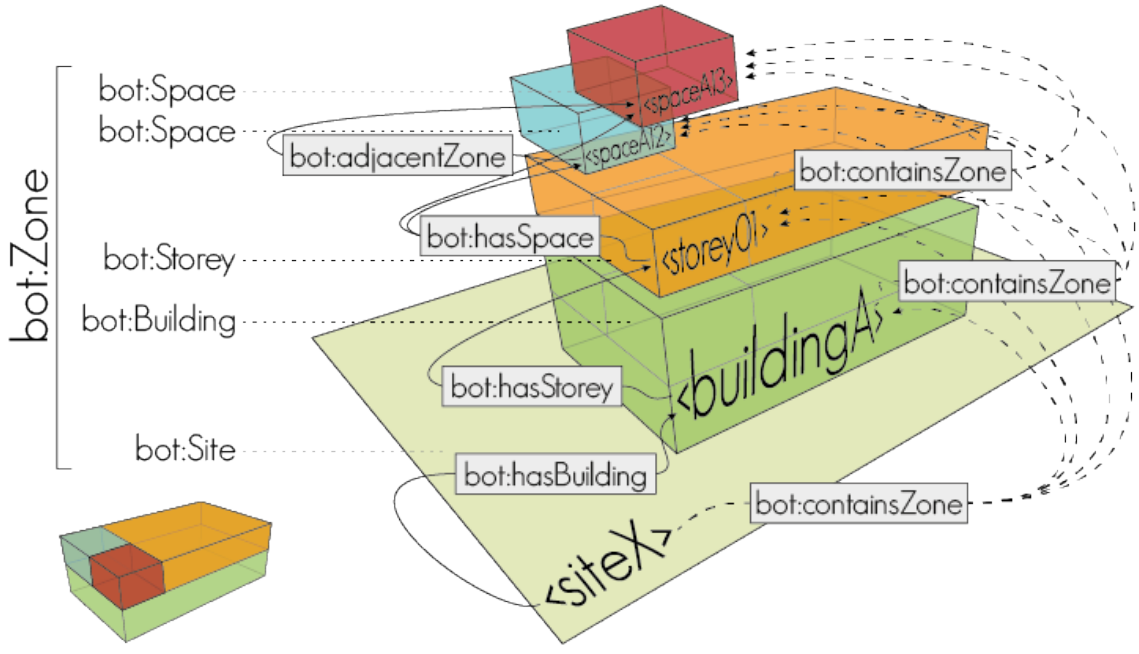



Abbildung 3.3: Zusammenhänge von *Zones* in BOT (RASMUSSEN et al., 2021)

Um nun topologisch aus diesen Konzepten ein Gebäude zu beschreiben fehlen noch die *Properties*. Zonen können sich mit anderen Zonen schneiden, ineinander liegen oder berühren. Für den Fall das zwei Zonen ineinander liegen gibt es noch *Sub Properties* für das Grundstück (*bot:hasBuilding*), das Gebäude (*bot:hasStorey*) und das Geschoss

(*bot:hasSpace*). Ebenso kann in einer Zone ein *Element* liegen, es kann diese überschneiden, oder nur berühren. Das Bezugsobjekt bleibt dabei die Zone. Anschließend kann auch ein Element noch ein Unterelement haben (*bot:hasSubElement*). Für die Verknüpfung mit Geometrie gibt es die Möglichkeit die *bot:Site* mit einem Basispunkt zu verknüpfen und jeder Zone und jedem Element eine 3D-Geometrie zuzuweisen.

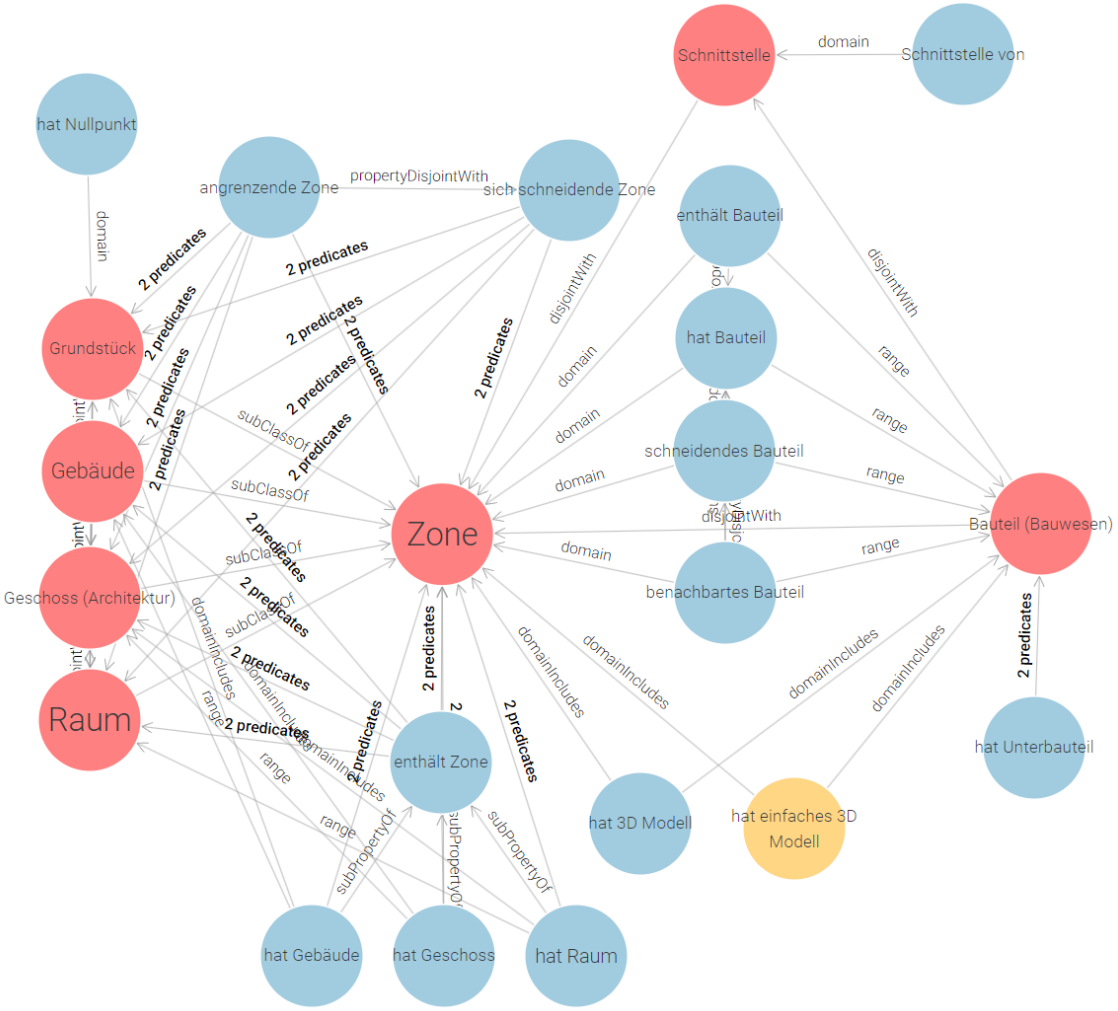



Abbildung 3.4: Graph der BOT Ontologie (RASMUSSEN et al., 2021). Dargestellt in GraphDB 2024b

Die Bezeichnung *2 Predicates* steht in der Abbildung für *rdfs:domain* und *rdfs:range*. Klassen sind rot dargestellt, *Object Properties* blau und *Datatype Properties* gelb.

Als Erweiterungen zu *BOT* gibt es unter anderen *PROPS* für die Repräsentation aller Eigenschaften aus *IFC*, *PRODUCT* für genauere Spezifikationen aller Elemente (RASMUSSEN et al., 2020, ORASKARI et al., 2024). Des weiteren sind Ontologien wie *OMG (Ontology for Managing Geometry)* und *FOG (File Ontology for Geometry Formats)* für die Verknüpfung und Darstellung von Geometrie (LBD-CG, 2022) kompatibel.

RASMUSSEN et al. (2020) merken bei ihrer Vorstellung von *BOT* bereits an, dass die Zonen auch als Träger von Anforderungen genutzt werden, wie es in dieser Arbeit angedacht ist.

### 3.6.4 Weitere Ontologien

Weitere erwähnenswerte Ontologien sind z.B.:

- *OPM: Ontology for Property Management*: Ermöglicht die Dokumentation von Änderungen von Eigenschaften. Damit wird eine Versionierung ermöglicht und alte Stände bleiben dokumentiert. (RASMUSSEN & LEFRANÇOIS, 2018)
- *GeoSPARQL*: Die Ontologie ermöglicht die Repräsentation von Geometrie und geometrischen Eigenschaften sowie eine Georeferenzierung. (OGC, 2024)
- *BRICK*: Damit lassen sich Metadaten von Gebäuden und vor allem der darin enthaltenen Haustechnik abbilden. Dazu zählen unter anderem Produktdaten, Anforderungen, Zonen und die zugehörigen Bauteile selbst und deren Beziehungen. *BRICK* selbst ist kompatibel mit *BOT* und ist zudem Open Source. (BRICK, 2024)

## 3.7 SPARQL

Die *SPARQL Protocol and RDF Query Language (SPARQL)* ist die Abfragesprache zu *RDF*, welche eng verbunden mit *RDF* vom World Wide Web Consortium (W3C) entwickelt wird. Die aktuelle Version ist *SPARQL 1.1* (W3C, 2013). Während der Bearbeitung dieser Arbeit wurden die Entwürfe für *RDF 1.2* und *SPARQL 1.2* veröffentlicht (W3C, 2024). Da diese Version aber noch nicht offiziell eingeführt oder vollendet ist, wird darauf in dieser Arbeit nicht weiter eingegangen.

Einfache Abfragen sehen ähnlich aus wie in *SQL* (ISO/IEC 9075-1, 2023-06), aber auch in *SPARQL* findet man das Tripel-Prinzip von *RDF* wieder.

Algorithmus 3.5: Simple SPARQL Abfrage

```
1 SELECT *
2 WHERE {
3     ?subject ?predicate ?object.
4 }
```

Die obige Abfrage zeigt die Abfrage aller Tripel in einem zugrunde liegenden Graph. Durch *SELECT* wird definiert, welche Variablen ausgegeben werden sollen. Variablen werden durch ein „?“ oder „$“ gekennzeichnet und besagen, dass an dieser Position im Tripel kein konkreter Wert stehen muss. Das aus *SQL* bekannte *FROM* kann in einer Abfrage weggelassen werden, wenn der Speicherort des Graphen bereits beim Aufruf der Abfrage übergeben wird, ansonsten können hier lokale Dateien und auch Web-Links angegeben werden. Mit *WHERE* wird eine Abfrage eingeleitet von der man als Antwort Statements oder Teile davon aus dem Graph erhält.

Algorithmus 3.6: SPARQL Abfrage zu Wohnorten

```
# Beispieldatei personen.ttl

:Herbert :wohntIn    :Berlin.
:Anna    :wohntIn    :München.
:Ingo    :wohntIn    :München.
:Karl    arbeitetIn: :Berlin.

# ------------------------------------------------------------------
PREFIX pers: <personen.ttl>

SELECT ?name
FROM <personen.ttl>
WHERE {
    ?name pers:wohntIn pers:München.
}

# ------------------------------------------------------------------
# Output (tabellarisch)
-------------
|    name   |
=============
| pers:Anna |
| pers:Ingo |
-------------
```

Dieses Beispiel ist bereits eine zielgerichtete Abfrage. Hier wird gezeigt, dass wie in *RDF* auch Präfixe vergeben werden können und bei *SELECT* konkrete Variabeln festgelegt werden können, damit nur diese im Ergebnis ausgegeben werden. Im *Triple Pattern* der Abfrage wurden nun statt ausschließlich Variablen, konkrete *RDF*-Elemente genutzt um zu beschreiben, dass man alle beliebigen Subjekte haben will, welche als Prädikat *pers:wohntIn* und als Objekt *pers:München* haben. Für komplexere Abfragen kann man auch mehrere Statements in einem *Triple Pattern* aneinander hängen. Das Ergebnis muss dann auch alle Tripel gleichzeitig zutreffen. Des weiteren kann man Abfrageteile als *OPTIONAL* deklarieren, wenn Ergebnisse auch ohne diesem Teil relevant sind (z.B. Abfrage von Mail-Adressen, optional auch nach einer Zweitadresse). Man kann zwei Abfragen mit *UNION* zu einem produzierten Ergebnis zusammenfassen, oder auch innerhalb einer Abfrage neue Unterabfragen erstellen. Damit man die Ergebnisse nicht ungefiltert hinnehmen muss, kann man diese z.B. mit *ORDER* sortieren, mit *DISTINCT* doppelte Einträge herausfiltern, mit *LIMIT* die Menge an ausgegebenen Datensätzen begrenzen oder mit *FILTER* die gefundenen Tripel nach Werten und nicht nur nach Übereinstimmung mit dem Pattern zu filtern. Zur Verdeutlichung folgt eine Abfrage, welche an ein Beispiel aus dem *SPARQL*-Lehrbuch von DuCharme (2013) angelehnt ist. Zur Vereinfachung wird auf Präfixe verzichtet.

Algorithmus 3.7: SPARQL Abfrage über Essenskosten. Grundlage ist eine Kombination von Beispielen aus DuCharme (2013) und einer Erweiterung um *:location*

```
# Beispieldatei essen.ttl

:meal1   :description "breakfast" ;
         :date "2011-10-14T08:53" ;
         :amount 6.53 .

:meal2   :description "lunch" ;
         :date "2011-10-14T13:19" ;
         :amount 11.13;
         :location "Canteen" .

:meal3   :description "dinner" ;
         :date "2011-10-14T19:04" ;
         :amount 28.30 .
# ---------------------------------------------------------------------
SELECT ?description ?location ?amount ((?amount * .2) AS ?tip) ((?
    amount + ?tip) AS ?total)
WHERE
{
  ?meal :description ?description ;
        :amount ?amount .
  OPTIONAL { ?meal :location ?location.}
}
ORDER BY DESC(?total)
# ---------------------------------------------------------------------
# Output (tabellarisch)

-----------------------------------------------------
| description | location   | amount | tip    | total  |
=====================================================
| "dinner"    |            | 28.30  | 5.660  | 33.960 |
| "lunch"     | "Canteen"  | 11.13  | 2.226  | 13.356 |
| "breakfast" |            | 6.53   | 1.306  | 7.836  |
-----------------------------------------------------
```

Hier sieht man, wie die Kosten für drei Mahlzeiten abgefragt und mit einem eher amerikanischem Trinkgeld versehen werden und anschließend der zu zahlende Gesamtbetrag je Mahlzeit ausgegeben wird. Dafür werden das Trinkgeld und der Gesamtbetrag als eigene Variable definiert und aus den abgefragten Werten berechnet. Da nur für das Mittagessen eine *Location* angegeben ist, muss diese Abfrage als *OPTIONAL* gekennzeichnet werden, sonst würden die anderen zwei Termine nicht in der Ausgabe erscheinen, da sie diese Bedingung nicht erfüllen können. Das Datum wird hier nicht abgefragt und erscheint deswegen auch nicht in der Abfrage. Anschließend werden die Ergebnisse nach absteigenden Gesamtkosten sortiert. Neben den gezeigten Berechnungen stehen in *SPARQL* auch Funktionen für mathematische Berechnungen, String-Analyse und -Bearbeitung und mehr zur Verfügung.

Neben reinen *SELECT*-Abfragen kann man z.B. mit *CONSTRUCT* aus den abgefragten Daten direkt neue Tripel generieren, oder mit *ASK* lediglich fragen, ob die Abfrage ein Ergebnis liefern würde (*True* oder *False*). Für weitere Informationen zu *SPARQL* kann

in der Dokumentation (W3C, 2013) oder z.B. in dem Lehrbuch von DuCharme (2013) nachgelesen werden.

## 3.8 Graphbasierte Regelprüfung

In Rasmussen et al. (2018) wurde bereits eine für diese Arbeit sehr passende Vorgehensweise für die Abfrage von Raumanforderungen auf Basis von *BOT* entwickelt. Dabei werden die Fragen abgedeckt, ob z.B. ein Raum seine Mindestflächenanforderung einhält, ob ein bestimmter Raumtyp oft genug im Modell vorhanden ist, oder ob zwei bestimmte Räume über *bot:adjacentZone* nebeneinander liegen. Dafür wird für jeden Raumtyp eine Unterklasse von *bot:Space* erstellt, welcher dann die geforderten Eigenschaften erhält. Für die Repräsentation der Werte wird die Ontologie *OPM (Ontology for Property Management)* verwendet, damit die Werte als Objekte und nicht nur als absoluter Datentyp verwendbar sind. In diesem Bericht wird *OPM* zudem um die Eigenschaft *opm:required* erweitert und die Werte von den definierten Raumtypen damit versehen. Somit lässt sich unterscheiden, ob ein Wert real vorkommt, oder ein Anforderungskriterium ist. Die Abfragen wurden dann in *SPARQL* ausgeführt.

Dieser Ansatz kann aber nicht verwendet werden, um Relationen zwischen zwei Räumen zu prüfen, da dort Pfade zwischen diesen gefunden werden müssen und diese Pfade liegen nicht explizit als Werte für einen Vergleich dieser Art vor. Allerdings kann dieses Abfrageschema genau dann angewendet werden, wenn rauminterne Eigenschaften, wie z.B. der Flächeninhalt, oder die Anzahl an Steckdosen geprüft werden.

# Kapitel 4

# Ontologie

In diesem Kapitel soll eine Ontologie hergeleitet werden, mit welcher die zu Beginn formulierten Fragestellungen für beliebige *IFC*-basierte Modelle beantwortet werden können. Dafür werden im Folgenden zunächst die grundlegenden Anforderungen aus den Fragen analysiert und anschließend daraus eine Ontologie formuliert. Ziel ist es, dass für Modelle möglichst wenige Einschränkungen erforderlich sind, um mit der Ontologie kompatibel zu sein. Auch wenn die Ausgangslage dieser Arbeit und die meisten nachfolgenden Beispiele Raumkonzepte von U-Bahnhöfen sind, soll die Ontologie möglichst allgemeingültig sein. Sie soll weiterhin widerspruchsfrei sowie möglichst dopplungsfrei und eindeutig sein. Das heißt, dass es im Gegensatz zu *IFC* möglichst nicht mehrere Wege geben soll, um den gleichen Sachverhalt darzustellen. Dies vereinfacht das Entwickeln von Abfragen, da weniger Repräsentationsvarianten berücksichtigt werden müssen und es erleichtert das Erlernen und Verstehen der Ontologie. Es müssen zudem beim Austausch von Daten weniger Abstimmungen getroffen werden, welche Varianten man wählt und zumindest bezüglich diesem Sachverhalt weniger projektspezifische Anpassungen an den eigenen Prozessen vorgenommen werden.

## 4.1 Analyse der Fragestellungen

Aus jeder Fragestellung ergeben sich notwendige Elemente, um diese Frage beantworten zu können. Diese Elemente müssen nicht neu sein, sondern können auch bereits durch andere Ontologien abgedeckt werden. Das Vorhandensein in einer anderen Ontologie ist kein grundsätzliches Ausschlusskriterium, da es sinnvoll sein kann, Sachverhalte effizienter zu modellieren und es sollten immer auch Verknüpfungen zu anderen Ontologien möglich sein.

**Frage 1:** *Sind alle geforderten Räume modelliert?* (Modellvollständigkeit)
Um tiefergehende Abfragen an den Graphen eines Gebäudes stellen zu können, muss man sichergehen können, dass selbiges vollständig abgebildet wird. Eine grundlegende topologische Gebäudestruktur zu modellieren ist, wie bereits erwähnt, das Grundkonzept von *BOT*. Mit *BOT* kann man die Zugehörigkeit von Räumen zu Gebäuden und Geschossen ausdrücken und dadurch bestimmen, was alles zum betrachteten Gebäude gehört und welches Element gegebenenfalls nicht verknüpft wurde. Aber es fehlt in *BOT* ein klarer Identifikator, wie z.B. ein Name oder ein Global Unique Indentifier (GUID), um ein Graph-Element klar identifizieren und rückverfolgen zu können, da eine *IRI* nicht zwingend eine interpretierbare Kennung darstellt, die eine Verbindung zum BIM-Modell ermöglicht. Diese Kennung sollte aber bereits in der *IFC*-Datei des Gebäudes vermerkt sein und damit

über *ifcOWL* und damit auch *PROPS* auffindbar sein. Eine direkte Verbindung zwischen Raum-Instanz und Kennung außerhalb der großen Menge an verknüpften Eigenschaften würde aber die Effizienz und Lesbarkeit steigern.

**Frage 2:** *Sind alle Räume erreichbar?* (Modellplausibilität)
Um zu ermitteln, ob alle Räume in einem Gebäude erreichbar sind, benötigt man zunächst eine Relation, welche diese Verbindung darstellen kann. Anschließend kann man die transitive Hülle des zugehörigen Graphen berechnen (MUNDANI, 2019). Sollte diese Hülle nicht alle Räume des Modells beinhalten, ist davon auszugehen, dass nicht alle Räume erreichbar sind. Dabei gilt die Annahme, dass ein Graph genau einem Gebäude entspricht. Wenn mehrere Gebäude im Graph enthalten sind oder ein Gebäude absichtlich geteilt ist (z.B. Doppelhaushälften), dann ist diese Trennung zu berücksichtigen und gegebenenfalls für jeden Teilbereich eine getrennte Analyse vorzunehmen. In *IFC* gibt es zwar keine direkte Eigenschaft, die festhält, welche Räume eine Tür verbindet. Aber diese Verbindungen können, wie in Abbildung 3.2, auch für Türen in den dort gezeigten Wänden ermittelt werden. Dadurch kann die Relation zwar auch über *ifcOWL* bestimmt werden, eine direkte Darstellung dieser Beziehung in der Ontologie wäre sinnvoll und würde den Berechnungsprozess wesentlich verkürzen.

Neben der reinen Möglichkeit Räume betreten zu können steckt in dieser Frage auch noch ein zweiter, die Zugänglichkeit betreffender Aspekt: Man stelle sich vor, man ist in einem Einkaufsmarkt und die Besuchertoilette liegt getrennt vom Verkaufsraum am Ende des Lagers. Wenn aber Kunden das Lager nicht betreten dürfen, ist die Toilette dann erreichbar? Deshalb sollte in der Ontologie auch berücksichtigt werden, dass es verschiedene Personengruppen (z.B. Mitarbeiter und Kunden) gibt, für welche gegebenenfalls Restriktionen bezüglich des Zugangs zu manchen Räumen gelten. Dies kann zur Folge haben, dass man neben der transitiven Hülle zur Prüfung der Plausibilität auch weitere für jede Personengruppe berechnen muss. Für diese Frage sind also zusammengefasst eine Relation für den vorhandenen Weg zwischen zwei Räumen, eine Klasse für Personengruppen und eine Relation für die Zugangsbeschränkung nötig. Dabei stellt sich die Frage, ob eine Zugangsbeschränkung für einen Raum gelten soll, oder für einen einzelnen Durchgang.

**Frage 3:** *Werden Mindestanforderungen eingehalten?*
In einem Anforderungskatalog wird üblicherweise nicht nur festgehalten, welche Räume benötigt werden, sondern auch, wie groß diese sein sollen und welche Mindestausstattung sie haben sollten. Im *Richtlinienkatalog U-Bahn (RLK)* (BAU-J & SWM, 2023) werden Anforderungen an die Raummaße, die Beschaffenheit der umschließenden Bauteile und die Raumausstattung gestellt. Dabei handelt es sich um interne Bedürfnisse, die zunächst keine Anforderung an einen anderen Raum stellen. Anforderungen an Gebäudeelemente (Decken, Böden, Wände) werden aufgrund ihrer Komplexität und spärlichen Auswirkungen auf Raumbeziehungen nicht weiter betrachtet. Aus den Beschreibungen von z.B. den Doppelböden in den Technikräumen im *RLK* folgen meist eher Anforderungen an andere Bauteile, als an Räume. Die grundlegenden geometrischen Eigenschaften (Grundfläche, Höhe, Volumen) beeinflussen auf topologischer Ebene zwar nicht die Beziehung

zu anderen Räumen, lassen sich aber einfach abbilden als interne Bedürfnisse. Aus der Raumausstattung als interne Anforderung resultieren aber meist Forderungen an Versorgung von außerhalb. Damit ist gemeint, dass ein gefordertes Telefon, Lichtschalter, Netzwerkanschlüsse, Lampen oder Heizung zwar Elemente sind, die nur den Raum selbst betreffen, aber die dahinterstehende Versorgung mit Strom, Netzwerk, Wasser usw. kommt meist von anderen Räumen und impliziert damit also in Konsequenz eine *Raumbeziehung*. Das heißt, zur Ableitung von sich aus internen Bedürfnissen ergebende Anforderungen sollte eine Ontologie möglichst auch interne Bedürfnisse von Räumen abbilden können, auch wenn es nicht das Kerngebiet der Ontologie ist. Da solche Anforderungen in *IFC* als *Property Sets* gespeichert werden, gibt es auch hier bereits theoretisch eine Darstellungsmöglichkeit in *PROPS* bzw. *ifcOWL*.

**Frage 4:** *Werden alle Bedürfnisse erfüllt?*
Diese Frage befasst sich mit den Beziehungen zwischen den Räumen, dem Prinzip von Nachfrage und Versorgung oder auch gleichwertige Relationen. Zu jedem Stromverbraucher bzw. dem Raum in dem dieser sich befindet muss eine Verbindung von einem Stromverteiler bzw. dem Hausanschluss führen und zu jedem Waschbecken mindestens ein Trink- und ein Abwasseranschluss. Dabei ergibt sich die Anforderung, dass eine Wasserleitung aus Sicherheitsgründen meistens nicht durch Elektrikräume führen darf (vgl. *RLK*). Die Relationen zwischen zwei Räumen müssen für jedes Bedürfnis getrennt dargestellt werden. Dies beruht darauf, dass Räume die Durchleitung von einzelnen Ressourcen verbieten, was wiederum ebenfalls dargestellt werden muss. Daraus ergeben sich gegebenenfalls wie schon für die Erreichbarkeit unterschiedliche Pfade zwischen zwei Räumen. Ein möglicher Pfad im Graph zwischen Ressourcenquelle und nachfragendem Raum zeigt, dass ein Bedürfnis erfüllt werden kann. Das gilt zunächst auf topologischer Ebene und garantiert z.B. nicht die Möglichkeit dort auch eine physische Leitung verlegen zu können. Wenn Bedürfnisse nicht die Gebäudetechnik oder andere physische Verbindungen repräsentieren sondern z.B. menschliche immaterielle Bedürfnisse, dann kann die Erfüllung wie eine Erreichbarkeitsanalyse aus Frage 2 geprüft werden. Es empfiehlt sich daher materielle und immaterielle Bedürfnisse getrennt zu deklarieren. Die Lösung dieses Problems wäre wie immer prinzipiell auch über *PROPS* möglich, aber die Graphpfade wären wesentlich länger und unübersichtlich gegenüber einer dedizierten Darstellung in einer Ontologie.

**Frage 5:** *Liegen Raumgruppen zusammen?*
Nach *RLK* sollen alle Räume eines Stellwerks zusammen auf einer Ebene liegen. In einer Wohnung sollten z.B. Küche, Wohn- und Esszimmer nicht weit voneinander liegen. Diese Beispiele treten aber auch gegebenenfalls geschossübergreifend auf. Um diese Forderungen prüfen zu können, ist lediglich ein geeigneter Identifikator, um die Raumart zu erkennen und die Lage der Räume vertikal und horizontal zueinander relevant. Ein passender Identifikator wäre der Raumtyp gegebenenfalls in Kombination mit einem eindeutigen Namen oder einer ID. Für die Umsetzung reicht *BOT* alleine diesmal nicht, da *bot:adjacentZone* an sich keine Unterscheidung beinhaltet, die trennt ob zwei Räume

nebeneinander oder übereinander liegen. Mit der Repräsentation von Name, Typ und ID verhält es sich wie bereits oben mehrfach beschrieben.

**Frage 6:** *Sind genügend Räume vorhanden?*
Neben der grundlegenden Existenz aller geforderten Raumtypen kann es auch vorkommen, dass von einem Typ mehrere Räume vorhanden sein müssen. Dabei sind für die Umsetzung mehrere Arten von Kriterien zu unterscheiden, da diese unterschiedlich gut topologisch abbildbar sind:

1. Räume, die pro Etage oder pro einer anderen Zone vorkommen müssen.
2. Räume, die durch eine Richtlinie oder vorherige Berechnungen in ihrer absoluten Anzahl bekannt sind.
3. Räume, deren Anzahl von der Menge oder dem Vorhandensein anderer Räume abhängt.
4. Räume, die anhand der Gebäudegeometrie platziert werden (z.B. alle 50 m, oder je 200 $m^2$).

Mittels *bot:containsZone* lässt sich überprüfen, ob erstere Raumtypen oft genug vorhanden sind. Die nächsten beiden Arten lassen sich trivial über den Vergleich zwischen gefundener Anzahl und berechneter Anzahl finden. Beim vierten Typ allerdings kommt die topologische Betrachtungsweise an ihre Grenzen. Während die Gesamtfläche errechnet werden kann, ist eine Bestimmung der Abstände oder andere präzise Lagerestriktionen ohne exakte Kenntnis der Geometrie nicht möglich. Ein Ansatz, wie den Mittelpunkt (oder einen anderen Punkt) eines Raumes in die Ontologie mit aufzunehmen, würde nur bei konvexen und kompakten Räumen funktionieren. Ein langer gebogener Flur würde durch so einen einzelnen Punkt nicht hinreichend repräsentiert werden können. Geht man aber davon aus, dass das Verhältnis von Gebäudegröße zu Raumgröße sehr klein ist, also die Räume eben kompakt sind, dann könnte man diesen Ansatz dennoch verfolgen. Dabei kann dann allerdings nicht mehr von einer Regelprüfung, sondern nur noch von einer Abschätzung gesprochen werden.

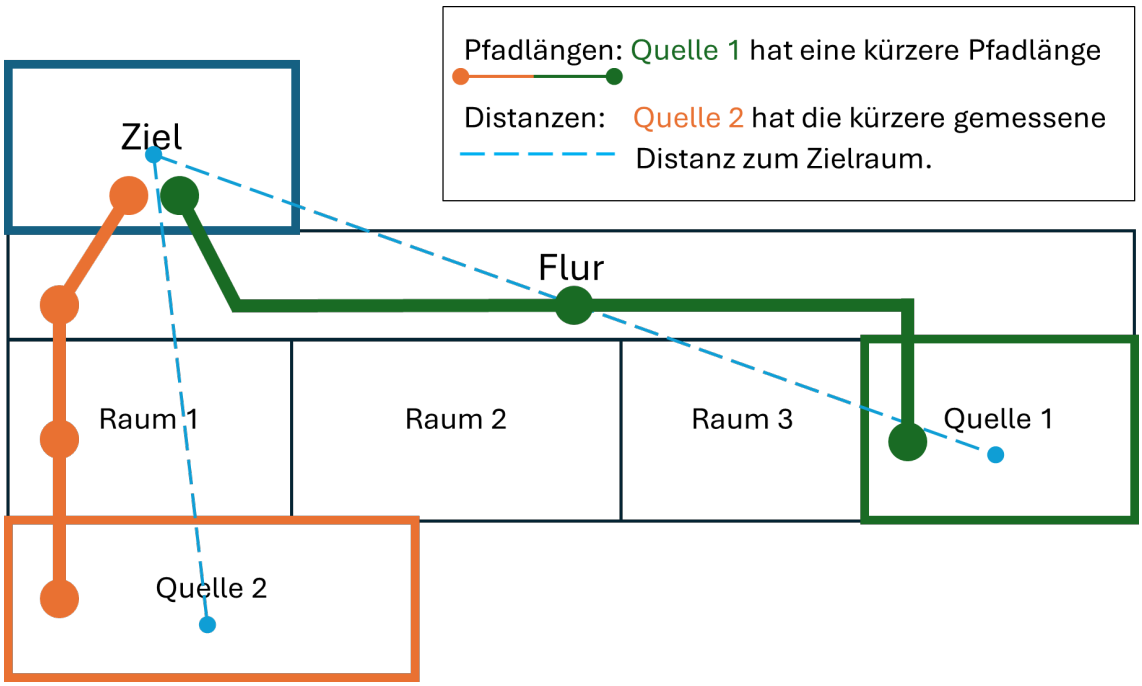



Abbildung 4.1: Durch den Vergleich der Pfadlängen oder der Distanzen ergeben sich zwei unterschiedliche kürzeste Wege

**Frage 7:** *Liegen thematische Räume günstig zueinander?*
Diese Frage enthält den subjektiven Begriff „günstig“ und unterscheidet sich von den anderen Fragen daher, dass es sich hierbei um eine Optimierungsfrage und keine strikte Prüfung handelt. Es muss also zunächst definiert werden, hinsichtlich welcher Kennwerte optimiert werden soll, da typischerweise mehrere Bereiche miteinander konkurrieren. Eine Optimierung hinsichtlich der Wünsche aller Parteien ist meist unmöglich. Stattdessen sollen sich aus der Ontologie Kennwerte ableiten lassen, mit denen ein Entwurf bewertet werden kann. Ein Beispiel wäre eine Minimierung von Leitungslängen für Wasserleitungen. Als Kennwert könnte die Anzahl der durchquerten Räume genutzt und als Hilfe die Abstände der Mittelpunkte der Räume verwendet werden, um Fehlinterpretationen zu verringern (z.B. Weg über zwei kurze Räume gegenüber einem Weg durch einen sehr langen Flur). Auch der Begriff „thematisch“ soll hier nicht näher definiert werden. Es sollen beliebige Kombinationen wählbar sein. Aus dieser Frage ergibt sich als Aufgabe für die Ontologie und vor allem die Abfragen, dass sie möglichst aussagekräftig sein sollten, um Vergleiche von verschiedenen Varianten zu ermöglichen.

Aus diesen Fragen wurden nun einige Anforderungen an die Ontologie zusammengetragen. Im nachfolgenden Abschnitt wird aus diesen Anforderungen eine mögliche Ontologie formuliert.

## 4.2 Konzeption der Ontologie

Bevor die wesentlichen Bestandteile der Ontologie erklärt werden, folgen ein paar grundlegende Informationen.

Die Ontologie wird *Room Analysis Ontology (RAO)* genannt mit dem Präfix *rao*. Eine konkrete öffentliche *URL* wird nicht festgelegt, da die Ontologie nicht öffentlich abgelegt wird. Für Testzwecke wird die *URI* „http://www.semanticweb.org/ontologies/RoomAnalysis/“ verwendet. Um die Ontologie universell einsetzen zu können, wird sie so simpel wie nötig gehalten. Konkrete Anforderungen und gegebenenfalls dafür nötige Subklassen, sollen vom Auftraggeber in einem eigenen *A-Box-Graph* formuliert werden. Der Modellierende muss dann dafür sorgen, dass sein *A-Box-Graph* des Gebäudes bezüglich der Bezeichnung und Struktur dem geforderten Schema angepasst ist (vgl. BIM-Merkmale in AIA (VDI 2552 BLATT 1, 2020-07)). Das Grundgerüst bildet die topologische Struktur von *BOT*. Die *RAO* ist also nicht ohne *BOT* verwendbar. Die meisten Relationen beziehen sich auf die Klasse *bot:Space*.

### 4.2.1 Klassen

Die in **Frage 1** geforderte Darstellung eines Raums wird durch *bot:Space* abgedeckt. Eine neue Klasse ist nicht erforderlich. Wäre es der einzige Berührpunkt mit *BOT*, so könnte man eigene Klassen für die Zonen implementieren und anschließend nur verlinken. Hier wird vereinfachend direkt auf *BOT* aufgebaut.

Als nächstes wurden Zugangsbeschränkungen für bestimmte Personengruppen gefordert. Mit der Klasse *Group* können hierfür einzelne Personengruppen bzw. Zugangsberechtigungen definiert werden. Auch Unterklassen sind hier möglich. Ein Beispiel wäre die Gruppe *Mitarbeiter*, welche sich gegenüber *Kunden* abgrenzt. Der *Hausmeister* ist ein *Mitarbeiter*, der zusätzlich zu den allgemeinen Mitarbeiterräumen auch in die Technikräume darf. Damit kann *Hausmeister* als Subklasse von *Mitarbeiter* definiert werden.

Für die Mindestanforderungen an die Räume selbst aus **Frage 3** gibt es die Klasse *InternalProperty*. Unter dieser Klasse werden alle physischen oder immateriellen Anforderungen an den Raum selbst gesammelt. Als Subklasse ist derzeit *GeometricProperty* standardmäßig definiert, es können aber auch für alle möglichen anderen Kategorien eigene Subklassen definiert werden.

**Frage 4** adressiert die Bedürfnisse. Jedes Bedürfnis muss als eine Ressource definiert werden, wenn es nicht im selben Raum, wo die Nachfrage besteht auch direkt befriedigt werden kann. Die Klasse *Resource* wird dafür in die Unterklassen *PhysicalResource* und *AbstractResource* unterteilt. Es wird also unterschieden zwischen physischen Ressourcen, welche tatsächlich fix mit dem anderen Raum verbunden werden müssen (z.B. Strom oder Wasser) und abstrakten Ressourcen wie z.B. Kaffee (als Getränk), welcher durch eine menschennutzbare Verbindung erreichbar sein muss. Weitere Unterklassen oder andere Unterteilung ab der Oberklasse sind möglich.

Die Aufnahme von expliziten Koordinaten wurde bereits in **Frage 6** für unsicher und nicht ausreichend befunden. Damit eine dort bereits erwähnte Abschätzung trotzdem möglich ist, wird eine Klasse *Point* definiert, welche aber nicht weiter spezifiziert wird. Ob ein Punkt angegeben werden soll und ob es ein Mittelpunkt ist, oder die Lage von Türen oder einer oder mehrere andere Punkte soll nicht vorgegeben werden.

Mit den aufgezählten Klassen lässt sich bereits ein Gebäude abbilden, aber es können noch keine Anforderungen gestellt werden. Der Klasse *Function* können alle Relationen, die für *bot:Space* als Raum konzipiert sind, zugeordnet werden. Eine Ausnahme bilden Punkte, da explizite Koordinaten nicht gefordert werden können. Wird z.B. eine Ressource einem Raum zugeordnet, ist es als *Ist*-Zustand bzw. die vorhandene Wahrheit zu verstehen und ist eine *Resource*-Instanz einer *Function* zugeordnet, dann stellt sie eine Anforderung dar. Jeder Raum, der diese Funktion zugeordnet bekommt muss diese Ressource bei einer Prüfung vorweisen können. Eine Instanz der Klasse *Function* kann ein Raumtyp sein, oder ein Teil davon. Als Beispiel wäre ein Raumtyp „Badezimmer“ und ein Teil dieses Typs, bzw. eine übergeordnete Kategorie wäre „Nassraum“, was dann lediglich festlegt, dass dort Frisch- und Abwasseranschlüsse vorhanden sein müssen. Damit *Function* die gleichen Relationen nutzen kann wie *bot:Space* muss *Function* formal auch eine Subklasse von *bot:Zone* werden.

Zuletzt gibt es noch die Hilfsklasse *RAOTopClass*, welche als Superklasse für alle neu definierten Klassen fungiert und diesen grundlegende Eigenschaften wie Name und Kennung vererbt. Darüber erhält jede Klasse die Möglichkeit ihren Instanzen eine GUID und einen Namen und eine beliebig wählbare Kennung zuzuweisen.

### 4.2.2 Object Properties

Zu den objektbezogenen Relationen zählt zunächst *accessibleFor*, um einem Raum oder einer Funktion eine zugangsberechtige Personengruppe zuzuordnen. Es können dabei beliebig viele Gruppen dem gleichen Objekt zugeordnet werden. Die Relation *hasProperty* für die Zuordnung von Instanzen der Klasse *InternalProperty* funktioniert exakt gleich. Punkte lassen sich dafür mit *hasPoint* nur tatsächlichen Räumen (*bot:Space*) zuweisen. Gleiches gilt für Funktionen mit *hasFunction*. Dabei darf ein Punkt, wie eine Gruppe, eine Funktion oder eine Eigenschaft auch mehreren Räumen gleichzeitig zugeordnet werden.

Für die Klasse *Resource* stehen drei Relationen zur Verfügung, mit welchen Bedürfnisse mit Räumen und Funktionen verknüpft werden können. Dabei wird mit *needs* ausgedrückt, dass ein Raum eine Ressource braucht und umgekehrt mit *provides*, dass ein Raum eine liefert. Zusätzlich gibt es noch die Möglichkeit mit *allowsPassingOf* festzulegen, welche Ressourcen durch einen Raum geleitet werden dürfen. In einem Elektroraum sollte z.B. verboten werden, dass Wasserleitungen hindurch laufen dürfen. Die Restriktion sollte ausschließlich für die Durchleitung gelten, damit nicht z.B. in einem Pumpenraum, in dem keine fremde Stromleitung durchgelegt werden darf, auch der Strom für die Pumpen verboten wird. Es empfiehlt sich bei vielen Ressourcen zusätzlich Parameter wie *all* festzulegen, um nicht immer die gesamte Ressourcenliste vollständig verknüpfen zu müssen, was

bei Änderungen in dieser zu Fehlern führen kann. Ebenso kann die Unterscheidung in materielle und immaterielle Ressourcen die Parametermenge verkleinern.

Zuletzt bleiben vier Erweiterungen der Relation *bot:adjacentZone* übrig. Während *bot:intersectsZone* und *bot:containsZone* ohne weiteres als Relation zwischen Räumen (und auch Funktionen) genutzt werden kann, ist *bot:adjacentZone* zu ungenau. Daher werden die drei Relationen *isAbove*, *isUnder* und *isNextTo* als Untertypen definiert, welche präzisere Zuordnung horizontal und vertikal zulassen. Wenn zusätzlich zur Nachbarschaft auch ein Zugang besteht, dann gibt es zusätzlich die Relation *hasAccessTo*. Da es sowohl für Zugänge über Türen, als auch Luken gelten soll, kann diese Relation nur Subtyp von *bot:adjacentZone* sein und nicht z.B. eine Präzision für *isNextTo*.

### 4.2.3 Datatype Properties

Für *bot:Zone* wird allgemein ein *Name* und eine GUID vorgesehen. *Bot:Zone* erhält zudem noch eine Raumnummer. Für alle Klassen der *RAO* wird über die *RAOTopClass* ein *Name*, eine GUID und eine nicht näher definierte Identifikationsnummer (*IdentNumber*) vererbt. Alle Tags dürfen nur einmal vergeben werden.

### 4.2.4 Zusammenfassung

Die *RAO* ist eine kompakte Grundlage für die Verknüpfung von Räumen über ihre Bedürfnisse und gleichzeitig Basis zur Definition von Anforderungen an die Räume. Sie bietet Erweiterungsmöglichkeiten durch Unterklassen, um beliebige Gebäude- und Regelstrukturen bestmöglich abbilden zu können. Es befinden sich daher auch keine vordefinierten Instanzen der einzelnen Klassen in der Ontologie.

Die Ontologie besteht im wesentlichen aus drei Teilen: einer präziseren Unterteilung von *bot:adjacentZone*, einem Set an neuen Relationen für *bot:Zone* und der Klasse *rao:Function* als theoretischer Raum, zur Bündelung von Anforderungen. Um die Übersichtlichkeit der folgenden Abbildung zu gewährleisten, wurden die Relationen zur *Raumfunktion* nicht dargestellt. Sie entsprechen im wesentlichen den Relationen von *Zone*, bis auf die zuvor erwähnten Ausnahmen. *Annotation Properties* und übergeordneten *OWL*-Eigenschaften werden ebenfalls nicht dargestellt. Detaillierte Beschreibungen der einzelnen Klassen und Relationen sind der Dokumentation zu entnehmen.

Für die Regeln zur Raumanzahl nach **Frage 6** ist es schwierig eine einheitliche, maschineninterpretierbare Relation zu entwickeln. Die unterschiedlichen Informationsanforderungen der dort erwähnten vier Arten (und die Liste ist nicht abgeschlossen) lässt sich nicht intuitiv in den Graph implementieren. Stattdessen sollten für jedes Regelwerk gezielt abgestimmte *SPARQL*-Abfragen definiert werden, welche diese Fälle individuell abfangen. Dies gilt auch für andere dynamische Parameter. Soll beispielsweise ein Lagerraum 5 m² je 30 m² Verkaufsfläche groß sein, dann muss dies über eine spezielle Abfrage geregelt werden. Man könnte eine „je etwas“-Relation implementieren, welche beliebige Klassen

als *range* nimmt. Dann wäre es aber dennoch nötig, gezielte Abfragen zu programmieren, da viele Kriterien unterschiedlich berechnet werden müssen.

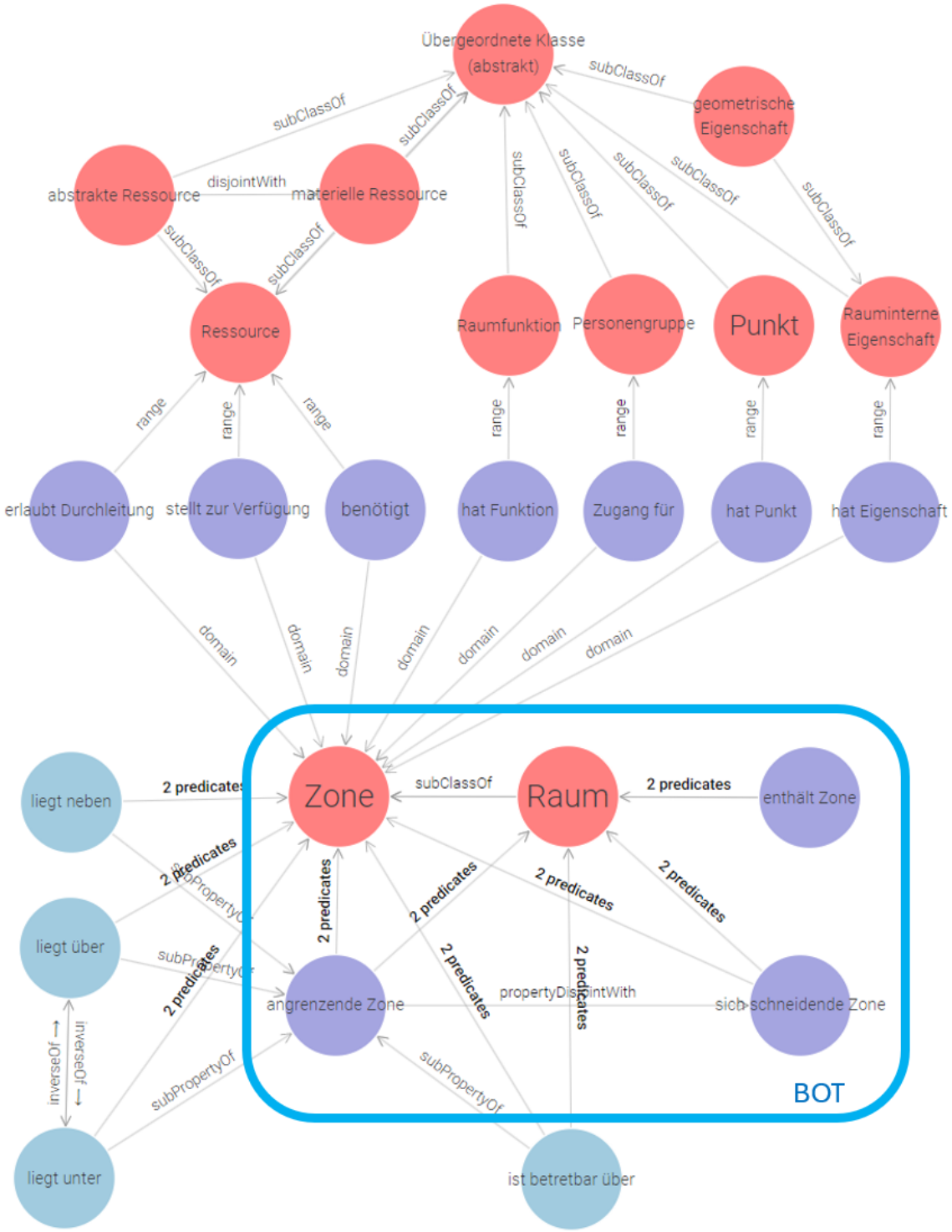



Abbildung 4.2: Graph der *Room Analysis Ontology* mit den Verknüpfungen zu *BOT*. Dargestellt in GraphDB 2024b

# Kapitel 5

# SPARQL Queries

Eine Ontologie ist erst nützlich, wenn man durch sie leichter oder überhaupt erst an Informationen gelangt. Daher werden in diesem Kapitel für jede Fragestellung beispielhaft *SPARQL*-Abfragen formuliert, mit welchen die jeweilige Frage auf eine Weise beantwortet werden kann.

Die Abfragen wurden rein in *SPARQL* geschrieben und in dem *Triplestore GraphDB* (ONTOTEXT, 2024b) getestet. Eine Kombination von *SPARQL* mit anderen Abfrage- oder Programmiersprachen ist möglich und kann sinnvoll sein. Eine Weiterverarbeitung von Abfrageergebnissen in *Python* wurde betrachtet, aber verworfen, um die Abfragen auch vollständig in *GraphDB* nutzen zu können und nicht an zwei unterschiedlichen Orten mit getrennter Software auswerten zu müssen. Alternativ müsste man einen *Triplestore* nutzen, der in ein *Pyhton*-Script integrierbar ist.

In den Abfragen wird das Präfix *inst:* für beispielhaft definierte Anforderungen und Instanzen verwendet. An dieser Stelle müssen anwendungsspezifische Klassen oder Individuals eingesetzt werden.

## 5.1 Abfragen zu den Fragestellungen

### 5.1.1 Abfrage 1: Modellvollständigkeit

Die Abfrage ob alle Räume im Graph bzw. Modell vorhanden sind, kann man auf mehrere Weisen verstehen. Die folgende Abfrage bezieht sich auf die Frage, ob für jede Funktion ein Raum vorhanden ist, unter der Annahme, dass jede in den Anforderungen definierte Funktion untergebracht werden muss. Nutzt man die Klasse *Function* stattdessen zur Beschreibung expliziter Räume, so lässt sich dann auch prüfen, ob alle gelisteten Räume vorhanden sind.

Algorithmus 5.1: Abfrage 1: Ist jede Funktion einem Raum zugeordnet?

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
3 SELECT *
4 WHERE{
5         ?name rdf:type rao:Function.
6     FILTER NOT EXISTS{?raum rao:hasFunction ?name}
7 }
```

Hier wird überprüft, ob für jede Instanz bzw. Subklasse von *rao:Function* eine Zuordnung zu mindestens einem Raum besteht.

### 5.1.2 Abfrage 2: Erreichbarkeit

Um zu überprüfen, ob alle Räume zugänglich sind, kann man testen, ob von einem beliebigen Raum aus alle anderen Räume über mindestens einen Pfad erreichbar sind. Dafür wird die Pfadsuche von ONTOTEXT (2024a) verwendet. Diese erlaubt mehrere Suchvarianten. In diesem Fall wird die Option „alle Pfade“ verwendet, um nicht nur einen kürzesten Pfad zum nächsten Raum zu bekommen, sondern alle möglichen Pfade zu allen Räumen. Dabei wird die Suche auf Kanten, als Prädikate vom Typ *rao:hasAccessTo* eingeschränkt. Sonst würden beliebige Relationen genutzt werden und die Suche könnte bei größeren Datenmengen sehr lange dauern. Als Rückgabewert wird hier lediglich eine Zahl ausgegeben. Der Wert *0* bedeutet, es gibt keinen unerreichbaren Raum im Modell. Größere Werte bedeuten, dass eine Anzahl an Räumen nicht vom ausgewählten Startraum erreichbar sind. Der Startraum kann wiederum beliebig gewählt werden.

Sollte man nicht nach der generellen Erreichbarkeit suchen, sondern nach der Erreichbarkeit für eine bestimmte Gruppe, so kann man die Pfadsuche auf Räume einschränken, welche diese Gruppe auch erlauben (in der folgenden Abfrage auskommentiert). Wenn statt der Anzahl an nicht erreichten Räumen die Namen dieser relevant sind, dann kann die Abfrage ähnlich wie Abfrage 7.4 umgeschrieben werden.

Algorithmus 5.2: Abfrage 2: Ist jeder Raum erreichbar?

```
PREFIX path: <http://www.ontotext.com/path#>
PREFIX inst: <file:/uploaded/generated/www.test.org#>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bot: <https://w3id.org/bot#>

SELECT (COUNT(?rooms)-1 - COUNT(?end) AS ?NichtErreichbar)
WHERE{
    {SELECT *
        WHERE{?rooms rdf:type bot:Space.}}
    UNION
    {SELECT DISTINCT ?end
    WHERE{
        VALUES (?src) {
            ( inst:RaumXXX)
        }
        SERVICE <http://www.ontotext.com/path#search> {
            <urn:path> path:findPath path:allPaths;
                       path:sourceNode ?src;
                       path:destinationNode ?dst;
                       path:pathIndex ?path;
                       path:resultBinding ?edge;
                       path:startNode ?start;
                       path:endNode ?end;
                       path:resultBindingIndex ?index.
            SERVICE <urn:path> {
                ?start rao:hasAccessTo ?end.
                # ?start rao:accessibleFor inst:Personengruppe.
                # ?end rao:accessibleFor inst:Personengruppe.
                }
            }
        }
    }
}
```

### 5.1.3 Abfrage 3: Mindestanforderungen

Für die internen Mindestanforderungen an die jeweiligen Räume müssen die *Soll*-Werte der verknüpften Funktionen mit den tatsächlich vorhandenen Raumeigenschaften verglichen werden. Die Vorlage von RASMUSSEN et al. (2018) unterstützt Vergleiche, ob ein Wert (z.B. ein Flächenmaß) exakt mit einer Anforderung übereinstimmt, oder in einem bestimmten Bereich liegt. Dabei dürfen nur die untersten Subklassen verglichen werden, da bei Elterklassen dann alle untergeordneten Instanzen mit allen verglichen werden, was nicht sinnvoll ist. Ausgegeben werden nur Zeilen, in denen ein Wert nicht eingehalten wird. Sollte die Klasse *InternalProperty* um weitere Kategorien erweitert werden, dann müssten diese ebenfalls im vorletzten Filter ausgeschlossen werden.

Algorithmus 5.3: Abfrage 3: Mindestanforderungen

```
#nach RASMUSSEN et al., 2018
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schema: <https://schema.org/>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?Raum ?Raumname ?Eigenschaft ?Funktion ?value ?reqVal
     ?matchViolated ?reqMin ?minViolated ?reqMax ?maxViolated
WHERE{
    ?Raum rao:hasFunction ?Funktion.
    ?Raum rao:hasProperty ?Maß.
    ?Raum rao:Name ?Raumname.
    ?Maß rdf:type ?Eigenschaft.
    ?Maß schema:value ?value.

    ?Funktion rao:hasProperty ?Referenz.
    ?Referenz rdf:type ?Eigenschaft.
    ?Eigenschaft rdfs:subClassOf+ rao:InternalProperty.

OPTIONAL {?Referenz schema:value ?reqVal}
OPTIONAL {?Referenz schema:minValue ?reqMin}
OPTIONAL {?Referenz schema:maxValue ?reqMax}

BIND( ?value != ?reqVal AS ?matchViolated )
BIND( ?value < ?reqMin AS ?minViolated )
BIND( ?value > ?reqMax AS ?maxViolated )

FILTER not exists{rao:InternalProperty rdfs:subClassOf+ ?Eigenschaft }
FILTER(?Eigenschaft != rao:InternalProperty && ?Eigenschaft != rao:
    GeometricProperty)
FILTER( ?matchViolated || ?minViolated || ?maxViolated )
}
```

### 5.1.4 Abfrage 4: Raumbedürfnisse

Für die Raumbedürfnisse muss wieder geprüft werden, ob ein Pfad existiert. Dieses mal geht es aber nicht um alle Wege von einem Raum aus, sondern es wird von jeder Quelle aus nur noch der jeweils kürzeste Weg zu allen nachfragenden Räumen gesucht. Dies verkürzt die Suchzeit. Der Pfad verläuft hier über aneinandergrenzende Räume, welche eine Durchleitung dieser Ressource auch erlauben. Der Zielraum ist von letzterem ausgenommen, da durch diesen keine Leitung mehr durchführen muss. Als Beispiel sei hier nochmal der Pumpenraum erwähnt, der selbst eine Stromversorgung benötigt, aber eine Durchleitung fremder Installationen verbietet. Das Individual *inst:XXX* stellt in den folgenden Abfragen eine beliebige Ressourcen-Klasse dar.

Die Abfrage wird hier in zwei Varianten gezeigt. Bei Variante 1 wird nur ausgegeben, wie viele Räume keinen Zugang zur Ressource haben. Es wird die Differenz berechnet zwischen der Anzahl an nachfragenden Räumen, gegenüber der Anzahl an Pfaden, die zu einzelnen nachfragenden Räumen möglich sind. Tauscht man die explizite Ressource durch die Variable *?res* aus (*inst:XXX* → *?res*), was die *VALUES*-Zuweisung überflüssig

macht und lässt diese final mit ausgeben, dann erhält man die Anzahl aller Räume mit nicht erfüllbaren Bedürfnissen, gestaffelt nach Ressource.

Algorithmus 5.4: Abfrage 4.1: Prüfung eines Raumbedürfnisses mit Rückgabe von nicht erreichten Räumen als Zahl

```
PREFIX path: <http://www.ontotext.com/path#>
PREFIX inst: <file:/uploaded/generated/www.test.org#>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX bot: <https://w3id.org/bot#>

SELECT (COUNT(?rooms) - COUNT(DISTINCT ?dst) AS ?NichtErfüllbar)
WHERE{
    {SELECT *
        WHERE{  ?rooms rao:needs inst:XXX.
                ?rooms rdf:type bot:Space.
        }}
    UNION
    {SELECT ?dst
    WHERE{
        VALUES (?res) {(inst:XXX)}
        ?src rao:provides inst:XXX.
        ?dst rao:needs inst:XXX.

        SERVICE <http://www.ontotext.com/path#search> {
            <urn:path> path:findPath path:shortestPath;
                       path:sourceNode ?src;
                       path:destinationNode ?dst;
                       path:pathIndex ?path;
                       path:resultBinding ?edge;
                       path:startNode ?start;
                       path:endNode ?end;
                       path:resultBindingIndex ?index.
            SERVICE <urn:path> {
                ?start bot:adjacentZone ?end.
                ?start rao:allowsPassingOf|rao:provides ?res.
                }
            }
        }
    }
}
```

Die zweite Variante gibt statt der Anzahl eine explizite Liste an Räumen aus, die nicht vom Quellraum aus erreicht werden konnten. Hier wurde die zuvor angesprochene Verallgemeinerung von einer Ressource auf alle bereits implementiert. Als Ergebnis bekommt man nun Kombinationen aus dem fehlerhaften Raum und seiner fehlenden Ressource. Statt dass die Summen der Ergebnisse verglichen werden, wird hier die Liste der Räume mit gefundenen Ressourcen-Pfaden von der Liste aller Räume mit Nachfrage subtrahiert.

Algorithmus 5.5: Abfrage 4.2: Prüfung eines Raumbedürfnisses mit Rückgabe von nicht erreichten Räumen als Liste

```
PREFIX path: <http://www.ontotext.com/path#>
PREFIX inst: <file:/uploaded/generated/www.test.org#>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bot: <https://w3id.org/bot#>

SELECT ?res ?dst
WHERE{
    ?dst rao:needs ?res.
    ?dst rdf:type bot:Space.
    MINUS{
    ?src rao:provides ?res.
    ?dst rao:needs ?res.

    SERVICE <http://www.ontotext.com/path#search> {
        <urn:path> path:findPath path:shortestPath;
                   path:sourceNode ?src;
                   path:destinationNode ?dst;
                   path:pathIndex ?path;
                   path:resultBinding ?edge;
                   path:startNode ?start;
                   path:endNode ?end;
                   path:resultBindingIndex ?index.
        SERVICE <urn:path> {
            ?start bot:adjacentZone ?end.
            ?start rao:allowsPassingOf|rao:provides ?res.
            }
        }
    }
}
```

### 5.1.5 Abfrage 5: Raumgruppen

Es folgt die Prüfung der geforderten topologischen Beziehungen. Sollen zwei Raumtypen aneinandergrenzen, dann müssen im Modell auch alle Räume mit diesen Funktionen aneinandergrenzen. Die *Abfrage 5* zeigt die Überprüfung für alle horizontalen Nachbarschaften. Für alle anderen Richtungen muss jeweils *rao:isNextTo* ausgetauscht werden. Auch die Relationen aus *BOT* sind nutzbar. Eine Kombination aller Relationen in einer Abfrage ist auch denkbar. Wie sich auch unterschiedliche Abfragen kombinieren lassen, wird in der nächsten Abfrage gezeigt.

Algorithmus 5.6: Abfrage 5: Prüfung von geforderten Nachbarschaften (hier nur horizontal)

```
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>

SELECT ?Raum ?Funktion ?Raum2 ?Funktion2
WHERE{
    ?Raum rao:hasFunction ?Funktion.
    ?Funktion rao:isNextTo ?Funktion2.
    ?Raum2 rao:hasFunction ?Funktion2.

    FILTER not exists{?Raum rao:isNextTo ?Raum2}
}
```

### 5.1.6 Abfrage 6: Raumanzahl

Abfragen zur Raumanzahl hängen sehr stark von den expliziten Anforderungen ab und sind daher nicht vollumfänglich bzw. allgemein abbildbar. In der gezeigten Abfrage wird beispielhaft die notwendige Anzahl von Toilettenräumen anhand von fiktiven Regeln geprüft. Die erste Regel zeigt wie man eine Anzahl in Abhängigkeit von der Grundfläche anderer Räume prüfen kann. Es wird festgelegt, dass je 100 m² Bürofläche eine Toilette vorhanden sein muss. Dafür werden die Grundflächen aller Büros aufsummiert und durch die Anzahl der Toiletten geteilt. Das Ergebnis wird als Kombination aus Beschreibungstext und boolescher Wert, ob die Anforderung eingehalten werten konnte, ausgegeben. Der zweite Teil der Abfrage beschäftigt sich damit, ob auf jeder Etage eine Toilette vorhanden ist. Gegebenenfalls müssen hier Ebenen rausgerechnet werden, welche keine Toilettenräume haben müssen oder können (z.B. Flachdach), aber für das Beispiel ist die Regel strikt. Auch hier wird das Ergebnis mit einem Beschreibungstext zurückgegeben, damit beide Teilabfragen in einer Tabelle erscheinen können und dabei unterscheidbar bleiben. Ein fiktives Ergebnis ist an das Ende der Abfrage angehängt.

Algorithmus 5.7: Abfrage 6: Beispiel: Kombinierte Abfrage, ob auf jeder Etage eine Toilette vorhanden ist oder je 100 m² Bürofläche

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bot: <https://w3id.org/bot#>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX inst: <file:/uploaded/generated/www.test.org#>
PREFIX schema: <https://schema.org/>

SELECT ?Typ ?Ergebnis
WHERE{{
    {SELECT (COUNT(?wcs) AS ?Toiletten)
    WHERE{
        ?wcs rdf:type bot:Space.
        ?wcs rao:hasFunction inst:WC.}}

    {SELECT ?Typ (SUM(?fläche) AS ?Gesamtbürofläche)
    WHERE{
        ?raum rdf:type bot:Space .
        ?raum rao:hasFunction inst:Büro.
        ?raum rao:hasProperty ?prop.
        ?prop rdf:type inst:Area.
        ?prop schema:value ?fläche.
        }GROUP BY ?Typ
        VALUES(?Typ){("Eine␣Toilette␣je␣100␣m2␣Bürofläche")}
        }
        BIND((?Gesamtbürofläche / ?Toiletten) <= 100 AS ?Ergebnis)
    }
UNION
    {SELECT ?Typ ((?AnzGeschosseGes - ?AnzGeschosseMit) = 0 AS ?
       Ergebnis )
    WHERE{
    {SELECT (COUNT(?floor2) AS ?AnzGeschosseGes)
        WHERE{
            ?floor2 rdf:type bot:Storey.
        }}
        {SELECT ?Typ (COUNT(?floor) AS ?AnzGeschosseMit)
        WHERE{
            ?wcs2 rdf:type bot:Space.
            ?wcs2 rao:hasFunction inst:WC.
            ?floor rdf:type bot:Storey.
            ?floor bot:hasSpace ?wcs2.
            }GROUP BY ?Typ
            VALUES(?Typ){("Eine␣Toilette␣je␣Geschoss")}
            }
        }
    }
}
# ---------------------------------------------------------------
# Output (tabellarisch)
-------------------------------------------------
|                  Typ                  | ?Ergebnis |
=================================================
| "Eine␣Toilette␣je␣100␣m2␣Bürofläche" | true      |
| "Eine␣Toilette␣je␣Geschoss"          | false     |
-------------------------------------------------
```

### 5.1.7 Abfragen zur Optimierung

Bezüglich der Optimierung ist es ähnlich schwer wie zuvor konkrete Abfragen zu definieren, da eine Optimierung stark vom Sachverhalt und der Priorisierung von Ressourcen abhängt. Um etwas optimieren zu können, benötigt man vergleichbare Kennwerte. Als Beispiel dafür wird im Folgenden die durchschnittliche Pfadlänge für Stromkabel berechnet. Der Wert zeigt an, wie viele Wände (bzw. Kanten) ein Kabel durchschnittlich zwischen Quelle und Zielraum durchqueren muss. Das ermöglicht zwar keine Aussagen zur Kabellänge aber zumindest Rückschlüsse auf die Anzahl an Durchbrüchen. Dieser Wert kann dann für verschiedene Varianten z.B. mit unterschiedlichen Lagen des Hauptverteilers berechnet und verglichen werden. Parallel dazu könnte man auch weitere Kennwerte definieren und berechnen, wie z.B. grobe Kabellängen über Raumpunkte (*rao:Point)* mit einem geschätzten Zuschlag, oder ob sich bei der Optimierung für den Strom eventuell die Situation bei der Wasserversorgung verschlechtert.

Die gezeigte Abfrage berechnet einzig einen Durchschnittswert aus den Pfadlängen, also über wie viele Kanten konnte ein Abnehmer von einem Versorgerobjekt durchschnittlich erreicht werden. Für die Auswertung der Kennwerte würden sich dann ggf. andere Programmiersprachen eignen, welche die Ergebnisse von mehreren Modellen sammeln, speichern und verarbeiten können.

Algorithmus 5.8: Abfrage 7: Beispielhafte Optimierungsgröße: Durchschnittliche Pfadlänge für Stromkabel.

```
PREFIX path: <http://www.ontotext.com/path#>
PREFIX bot: <https://w3id.org/bot#>
PREFIX rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/>
PREFIX inst: <file:/uploaded/generated/www.test.org#>

SELECT (ROUND(AVG(?dist)*10)/10 AS ?Average)
WHERE{
        ?src rao:provides inst:Strom.
    ?dst rao:needs inst:Strom.

    SERVICE path:search {
        <urn:path> path:findPath path:distance;
           path:sourceNode ?src;
           path:destinationNode ?dst;
           path:startNode ?start;
           path:endNode ?end;
           path:distanceBinding ?dist.
        SERVICE <urn:path> {
                ?start bot:adjacentZone ?end.
            ?start rao:allowsPassingOf inst:Strom.
        }
    }
}
```

## 5.2 Zusammenfassung

Die dargestellten Abfragen zeigen, dass die Ontologie geeignet ist, die ursprünglich aufgestellten Fragen zu beantworten. Die Abfragen können und müssen teilweise auch auf konkrete Anwendungsfälle zugeschnitten werden und die Ergebnisse können auf verschiedene Weisen dargestellt werden. Die Fragestellungen können allein mittels *SPARQL* vollständig beantwortet werden, aber auch eine Kombination mit anderen Sprachen, wie bei der Optimierung kann nützlich sein.

Um die Funktionsweise der Abfragen zu demonstrieren werden in Kapitel 7 explizite Beispiele, bestehend aus Modell und Anforderungen durchgearbeitet. Dafür müssen aber zunächst Algorithmen entwickelt werden, welche die Informationen aus *IFC*-Dateien in einen *RDF*-Graphen mit dieser Ontologie überführen.

# Kapitel 6

# Umsetzung

Um nun BIM-Modelle in einen Graphen umwandeln zu können werden einige Tools benötigt. Die einzelnen Informationen, welche mit der Ontologie repräsentiert werden sollen, lassen sich unterschiedlich leicht aus den *IFC*-Daten extrahieren. Für die Bestimmung der Raumnachbarschaften ist ein neuer Algorithmus nötig und für die Erkennung von Zugängen können bekannte Konzepte genutzt werden und die Informationen über die Räume können ebenso über bewährte Wege ermittelt werden. Damit die Informationen aus der *IFC*-Datei ausgelesen werden können, müssen diese, sofern sie nicht zum *IFC*-Standard gehören, extra im Autorenprogramm definiert und anschließend auch exportiert werden. Um alle Informationen für die Ontologie zu sammeln, wird ein Python-Script erstellt, welches eine *IFC*-Datei einliest und eine *Turtle*-Datei ausgibt.

## 6.1 Nachbarschaftsalgorithmus

Da einerseits die Umsetzung der *BOT*-Relationen *adjacentZone, intersectsZone, containsZone* im aktuellen Stand des LBD-Konverters fehlt (BONDUEL et al., 2018) und für die *RAO* gleichzeitig präzisere Unterscheidungen nötig sind, war es erforderlich in Zuge dieser Arbeit dafür einen eigenen Konverter zu schreiben. Dafür werden im Folgenden zwei Algorithmen beschrieben, welche die Nachbarschaft von zwei Räumen horizontal bzw. vertikal bestimmen können. Zuerst werden die jeweiligen Überlegungen schrittweise hergeleitet und anschließend wird die tatsächliche Umsetzung beschrieben.

In LANGENHAN et al. (2013) wird für die Bestimmung der Nachbarschaft zweier Räume die bereits in Abschnitt 3.2 gezeigte *IfcRelSpaceBoundary*-Beziehung (BUILDINGSMART INTERNATIONAL, 2023) verwendet. Also wenn eine Wand die Raumbegrenzung (*IfcRelSpaceBoundary*) für zwei Räume ist, dann liegen diese nebeneinander. Dies ist allerdings eine starke Vereinfachung, denn dieses Prinzip funktioniert bei weitem nicht immer eindeutig und wird deshalb in dieser Arbeit nicht verwendet. Ein einfaches Beispiel wäre, dass an eine lange Flurwand mehr als zwei Räume angrenzen können und rein aus der Beziehung heraus nicht ersichtlich ist, welche davon nun tatsächlich nebeneinander liegen. Eine Stückelung der Wände, sodass sie stets nur zwei Räume trennen, mag in der Theorie funktionieren, würde allerdings den realen Modellierungsablauf und die Nutzung beliebiger Modelle stark erschweren. Stattdessen wurde ein Ansatz gewählt, bei dem die Lage der Räume zueinander über deren Geometrie mathematisch berechnet wird.

### 6.1.1 Überlegung zur Bestimmung horizontaler Nachbarschaften

Um angrenzende Räume auch ohne Hilfe des *IFC2LBD*-Konverters (ORASKARI et al., 2024) zu erkennen, wird hier ein Verfahren gewählt, dass auf vergrößerten Geometrien basiert. Die Grundidee beruht darauf, dass wenn zwei Räume nebeneinander liegen, diese höchstens eine Wanddicke voneinander getrennt sind. Wenn man also die Geometrien der Räume in alle Richtungen um mindestens die halbe Wanddicke vergrößert, dann überschneiden sich angrenzenden Räume und man kann dies über gängige Kollisionsprüfungen berechnen. Wenn sich zwei Räume dann in ihrer vergrößerten Form überschneiden, aber nicht in ihrer regulären, dann kann man mit Ausnahmen davon ausgehen, dass die Räume nebeneinander liegen. Als *nebeneinander* gelten in diesem Fall Räume, die mindestens ein paralleles Kantenpaar besitzen, welches maximal eine vorgegebene Distanz auseinander liegt. Diese Distanz definiert wie nah sich Räume sein müssen, damit sie als benachbart gelten. Der Wert sollte etwas größer sein, als die dickste Wand, um auch die ungünstigsten Fälle abzudecken, aber gleichzeitig nicht schmale Räume zu überspringen.

Von der *IFC*-Datei aus sind die Räume als Polygon auslesbar und in der konvertierten LBD-Fassung existiert bereits für jedes Objekt eine Axis Aligned Bounding Box (AABB). Diese sind aber auch aus den Polygonpunkten berechenbar. AABBs sind entlang der Koordinatenachsen ausgerichtete, objektumschließende Rechtecke von den minimalen X- und Y-Werten der Raumgeometrie, bis zu deren Maximalwerten und eignen sich für schnelle Kollisionsprüfungen (ERICSON, 2005). Es folgt eine schrittweise Annäherung an diese Lösung:

**Schritt 1: Betrachtung der Bounding Boxen**

Zu Beginn der Überlegung liegen nur die Raumumrisse als plane Flächen bzw. Polygone vor. Aus den Koordinaten der Polygonpunkte lassen sich vereinfachende Bounding Boxen berechnen. Die einfachste Form ist die oben genannte AABB, welche sich aus den Minimal- und Maximalwerten der Polygonkoordinaten berechnet und dadurch Kollisionsberechnungen von sonst teilweise komplexen Objekten stark vereinfacht. Diese Methode ist zwar ungenau, ermöglicht aber die starke Reduzierung der detaillierter zu berechnenden Raumpaarungen. Über diese Bounding Boxen lassen sich aufgrund ihrer gröberen Umfassung der Räume bereits erste Überschneidungen mit den Boxen anderer Räume feststellen, was diese zu möglichen Kandidaten für genauere Betrachtungen ihrer Beziehung macht. Allerdings hängt die Erkennung davon ab, wie die Räume geformt sind und wie sie bezüglich der Koordinatenachsen liegen. In Abb. 6.1 ist zu sehen, dass Überschneidungen mit dieser Methode, zukünftig *AABB vs. AABB* genannt, eher zufällig sind und da das Modellkoordinatensystem zum Zeichnen üblicherweise möglichst parallel zu den Gebäudeachsen gelegt wird (wie links im Bild), sind auch wenige Ergebnisse zu erwarten.

Da sich Überschneidungen von AABB sehr simpel berechnen lassen, wird dennoch an diesem Prinzip zunächst festgehalten. Der Algorithmus für die Berechnung besteht

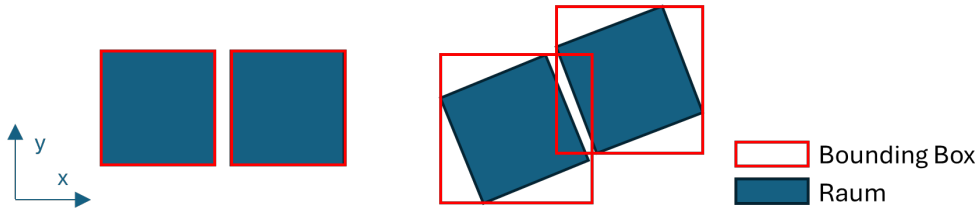



Abbildung 6.1: Methode *AABB vs. AABB*. Überschneidungen von standardmäßigen Bounding Boxen in Abhängigkeit der Raumausrichtung.

lediglich aus einer Abfrage für jede Dimension (beliebig erweiterbar), ob der Minimalwert der einen Box kleiner ist als der größte der anderen Box. Ist dieses Kriterium für alle Dimensionen gleichzeitig erfüllt, dann überschneiden sich die Bounding Boxen, sonst nicht. Daraus lässt sich schließen, dass zwei Objekte sich entweder ganz sicher nicht schneiden, da sich die vollständig umschließenden Boxen nicht schneiden, oder dass eine Überschneidung möglich wäre und eine genauere Untersuchung durchzuführen ist.

Algorithmus 6.1: Codebeispiel für die Überschneidung zweier AABBs. Basiert auf ERICSON (2005)

```
1 Boolean AABB_Intersection(AABB a, AABB b)
2 {
3     if (a.max_X < b.min_X || a.min_X > b.max_X) return false;
4     if (a.max_Y < b.min_Y || a.min_Y > b.max_Y) return false;
5     return true;
6 }
```

Um nun in jedem Fall und nicht nur bei einem bestimmten Winkel die Nachbarräume bestimmen zu können, kann man die AABBs der Räume jeweils so weit in jede Richtung vergrößern, dass die Boxen durch Wände hindurch in die nächstgelegenen Räume ragen (Methode *erweiterte AABB vs. erweiterte AABB*). Der Vergrößerungsfaktor sollte dabei mindestens der halben Dicke der dicksten Wand entsprechen, damit sich die AABBs auch im ungünstigsten Fall schneiden. Damit können noch mehr Räume als potenziell nebeneinanderliegend erkannt werden. Dass auch dieser Ansatz noch nicht fehlerfrei ist, zeigt die folgende Grafik.

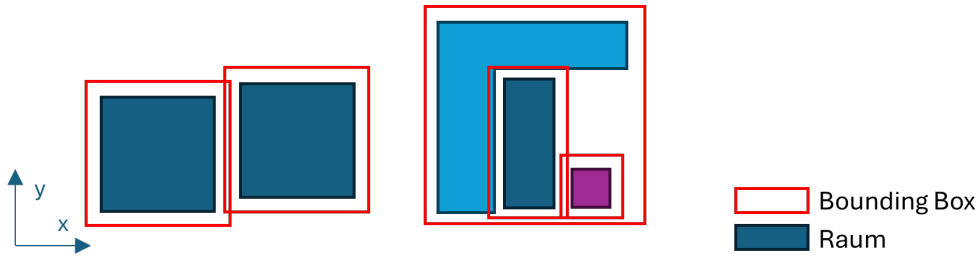



Abbildung 6.2: Methode *erweiterte AABB vs. erweiterte AABB*. Darstellung der Verbesserungen gegenüber *AABB vs. AABB* und Grenzen der Methode.

In der obigen Skizze ist links zu sehen, dass durch die vergrößerten AABBs auch achsparallele Räume durch deren Überschneidung als Nachbarn erkannt werden können. Sobald aber ein Raum (z.B. ein langer gebogener Flur) durch seine Form eine Bounding Box

erzeugt, welche so viel Fläche um den Raum herum miteinschließt, dass andere Räume, welche nicht an den eigentlichen Raum angrenzen, vollständig darin liegen können, ist auch diese Methode nicht mehr aussagekräftig. Im rechten Teil der Abbildung sieht man so einen Fall. Die Bounding Box des L-förmigen hellblauen Raums umschließt die beiden anderen Räume inklusive deren Bounding Boxen. Der oben gezeigte Algorithmus 6.1 erkennt eine Überschneidung für beide Räume mit dem hellblauen Raum. Da allerdings nur der dunkelblaue Raum tatsächlich an den hellblauen angrenzt und der violette nicht , muss eine Möglichkeit gefunden werden diese beiden Fälle zu unterscheiden.

Dazu gibt es zwei Möglichkeiten. Entweder man führt die Überlegung auf Basis der AABBs fort, oder man betrachtet direkt die Polygone. Eine Untersuchung hinsichtlich der Effizienz und des Rechenaufwands findet im Rahmen dieser Arbeit nicht statt.

**Variante 2.1: Polygone und Bounding Boxen**

Im obigen Beispiel lässt sich beobachten, dass zwar der violette Raum in der erweiterten Bounding Box des hellblauen Raumes liegt, aber nicht umgekehrt. Das trifft allerdings auch noch zu, wenn die AABB des hellblauen Raums nicht vergrößert wäre. Daraus folgt die Überlegung die Kollisionsprüfung asymmetrisch zu gestalten. Es sollen nun nicht mehr alle Räume gleichzeitig mittels erweiterten Bounding Boxen gegeneinander geprüft werden, sondern immer nur ein Raum mit erweiterter AABB gegen die reale Geometrie der anderen Räume. Die Methode wird nachfolgend *AABB vs. Polygon* genannt. Die zuvor entwickelte Stufe des Algorithmus (*erweiterte AABB vs. erweiterte AABB*) kann allerdings zur groben Vorauswahl der näher zu betrachtenden potenziellen Nachbarschaftsbeziehungen genutzt werden. Diese Stufe kann bereits schnell viele Überlagerungen sicher ausschließen, aber eben noch keine verlässlichen Zusagen liefern. Dadurch werden die rechenaufwändigeren Polygon-Berechnungen auf einen kleineren Kreis von potenziellen Nachbarschaften begrenzt, was wiederum die Rechenzeit senkt.

Nach der *AABB vs. Polygon*-Methode gelten zwei Räume als benachbart, wenn der Überschneidungstest in beide Richtungen erfolgreich war. Das bedeutet, dass sowohl die Bounding Box von Raum *1* das Polygon der unvergrößerten Grundfläche von Raum *2* schneidet oder beinhaltet, als auch andersherum. Die Vergrößerung der Bounding Box sollte nun der gesamten Distanz entsprechen, innerhalb der ein Raum als angrenzend gelten soll. In der folgenden Abbildung ist zu sehen, wie der Test für den hellblauen gegen den violetten Raum negativ ausfällt, da zwar der violette Raum in der Bounding Box des hellblauen Raumes liegt, aber die erweiterte Bounding Box des violetten Raumes den hellblauen Raum nicht schneidet. Für den zweiten Fall *„Hellblau gegen Dunkelblau“* ergibt die Prüfung in beide Richtungen ein positives Ergebnis, weshalb davon auszugehen ist, dass beide Räume tatsächlich aneinandergrenzen.

Der Berechnungsalgorithmus für den Überschneidungstest kann weder einzig auf dem Prinzip *„Eckpunkt der einen Fläche liegt innerhalb der anderen“* basieren, noch kann man sich rein auf die Schnittpunkte von einzelnen Kanten verlassen. Bezüglich des ersten Punktes kann es der Fall sein, dass alle Eckpunkte eines Raumes außerhalb der Bounding Box des anderen Raumes liegen und diese Box gleichzeitig mit keinem

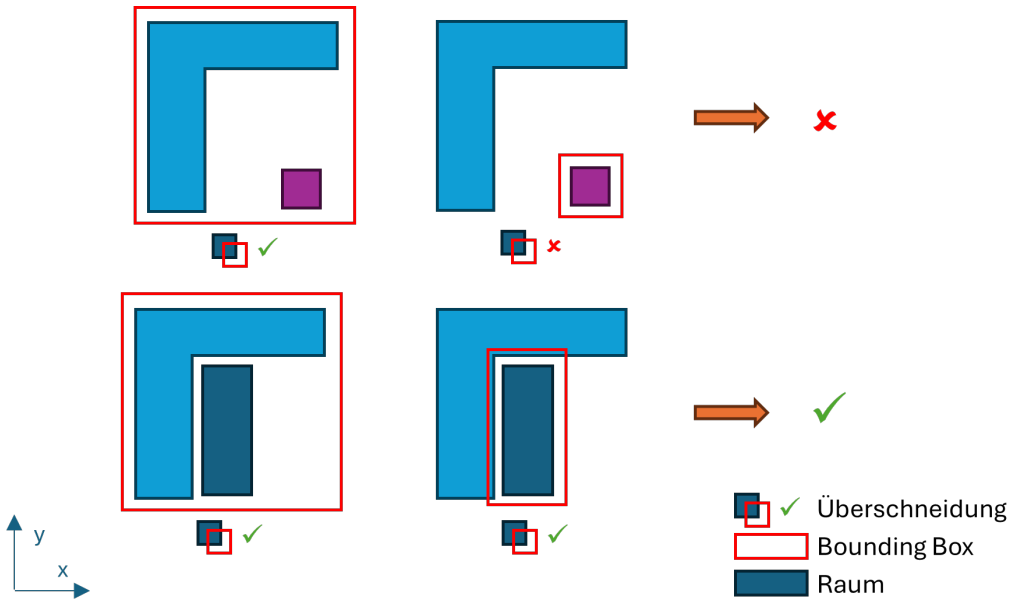



Abbildung 6.3: Methode *AABB vs. Polygon*. Erst wenn die Methode in beide Richtungen funktioniert, kann man sagen, dass zwei Räume nebeneinander liegen.

Eckpunkt in diesem Raum liegt (siehe Beispiel unten). Der Punkt mit den fehlenden Schnittpunkten der Geometrien, weil diese ineinander liegen, ist bereits in der obigen Abbildung zu erkennen, da beide kleinen Räume vollständig in der Bounding Box des hellblauen Raumes liegen.

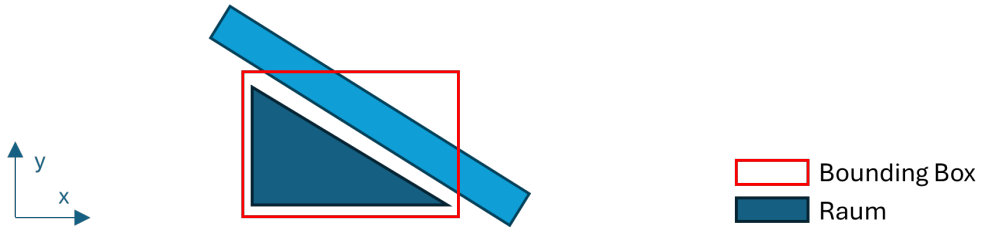



Abbildung 6.4: Kollision eines Raumes mit einer AABB, ohne dass ein Punkt oder der gesamte Raum innerhalb liegt.

Eine Kombination aus beiden Prinzipien würde eine Lösung bringen, erhöht aber auch den Rechenaufwand. Anschließend sollten nur noch tatsächlich *nah* aneinander liegende Räume in der Lösung vorhanden sein. Was *nah* ist, wird dabei durch das Maß der Erweiterung der Bounding Box bestimmt. Allerdings sind nun auch noch Räume in der Lösung, welche nur über eine Ecke aneinander grenzen. Um diese herauszufiltern, muss auf die Lösung noch eine weitere Prüfstufe angewendet werden. In diesem Schritt werden alle gefundenen Nachbarschaften erneut berechnet. Nur wird dieses Mal die AABB viermal in je eine Koordinatenrichtung erweitert, damit reine Eckbeziehungen nicht mehr erkannt werden. Sollte keine der Varianten eine Kollision erkennen, ist davon auszugehen, dass die Räume nur nah zusammen liegen, aber nicht nebeneinander in dem Sinne, dass sie sich eine Raumseite mindestens teilweise teilen. Diese Variante funktioniert jedoch nur gut für achsparallele Räume.

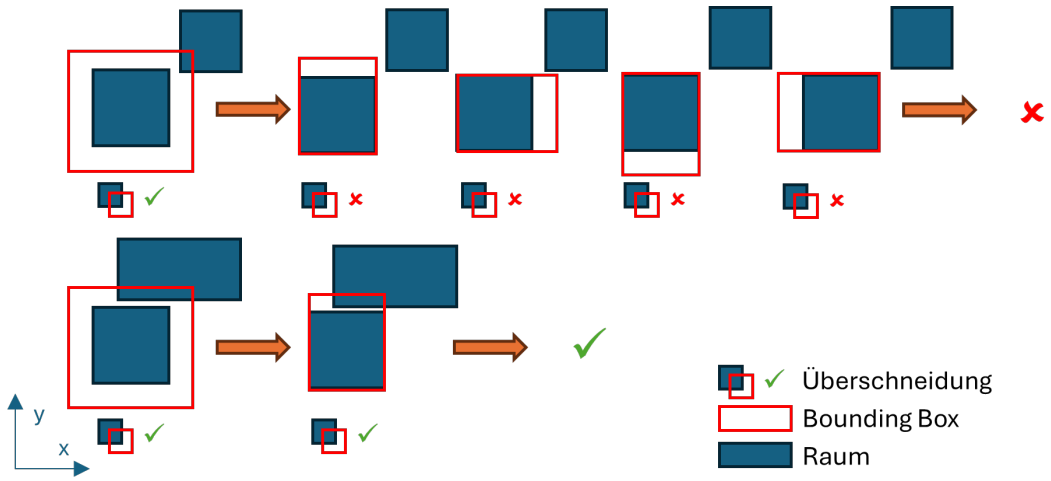



Abbildung 6.5: Erweiterung der AABB in jeweils nur eine Richtung schließt Eckbeziehungen aus.

Anschließend ist noch zu prüfen, ob beide Räume tatsächlich parallele nah aneinander grenzende Kanten haben. Dies wird in der zweiten Variante mitbetrachtet.

#### Variante 2.2: Erweiterte Polygone

Eine Alternative zur ersten Variante wäre, statt die Bounding Box zu erweitern, jeweils eine Kante des Polygons als eine Art Oriented Bounding Box (OBB) zu erweitern und dann diese Box für den Test zu verwenden. Da eine OBB nicht mehr an den Achsen ausgerichtet ist, ist der Rechenaufwand etwas höher, dafür ist die Berechnung aber tendenziell genauer (ERICSON, 2005). Weil die Box für jede Kante einzeln erzeugt wird, ist diese Variante sogar noch etwas genauer als eine normale OBB.


Abbildung 6.6: Genauigkeit von modifizierten *Orientend Bounding Boxen* gegenüber der bisher verwendeten *erweiterten AABB* (gestrichelt).

#### Schritt 3: Prüfung naher Räume auf Parallelität

Da zusätzlich die Option besteht, dass Räume nur über eine Ecke aneinandergrenzen, sollte man auch über die Lage der Polygonkanten zueinander den Raumabstand bestimmen. Es wird hierfür angenommen, dass Wände und sonstige Raumabgrenzungen eine konstante Dicke haben, wodurch die angrenzenden Kanten von benachbarten Räumen parallel verlaufen. Unter dieser Annahme gibt es also für benachbarte Räume jeweils mindestens eine Kante, welche beide zueinander parallel verlaufen und deren Abstand zueinander kleiner ist, als die Erweiterung der Bounding Box.

Eine gewisse Toleranz bezüglich der Parallelität von Kanten und dass diese auch ohne Abstand nebeneinander liegen können, muss bei der Umsetzung berücksichtigt werden. Ebenso muss über eine klassische Kollisionsprüfung sichergestellt werden, dass sich betrachtete Räume nicht überschneiden, oder ineinander liegen.

Änderungen des Grundrisses über die Höhe der Räume, oder Räume mit Höhenversatz wurden in dieser Überlegung explizit ausgeklammert. Es ist auch davon auszugehen, dass viele Spezialfälle an Raumgeometrie-Kombinationen nicht korrekt erkannt werden. Zudem hat sich herausgestellt, dass auch die Reihenfolge der Teilschritte eine Auswirkung darauf hat, wie viele Nachbarschaften erkannt werden. Dazu folgt mehr in Abschnitt 6.1.3.

### 6.1.2 Überlegung zur Bestimmung vertikaler Überlagerungen

Um vertikale Raumnachbarschaften zu finden, kann ähnlich vorgegangen werden. Eine vertikale Nachbarschaft lässt sich am einfachsten so beschreiben, dass sich die Grundrisse zweier Räume auf angrenzenden Ebenen überlagern. Im Gegensatz zur Nachbarschaft in der Ebene müssen hier zwei Fälle unterschieden werden, denn es kann notwendig sein zu wissen, ob ein Raum über oder unter dem anderen liegt. Es könnte deshalb der *erweiterte*

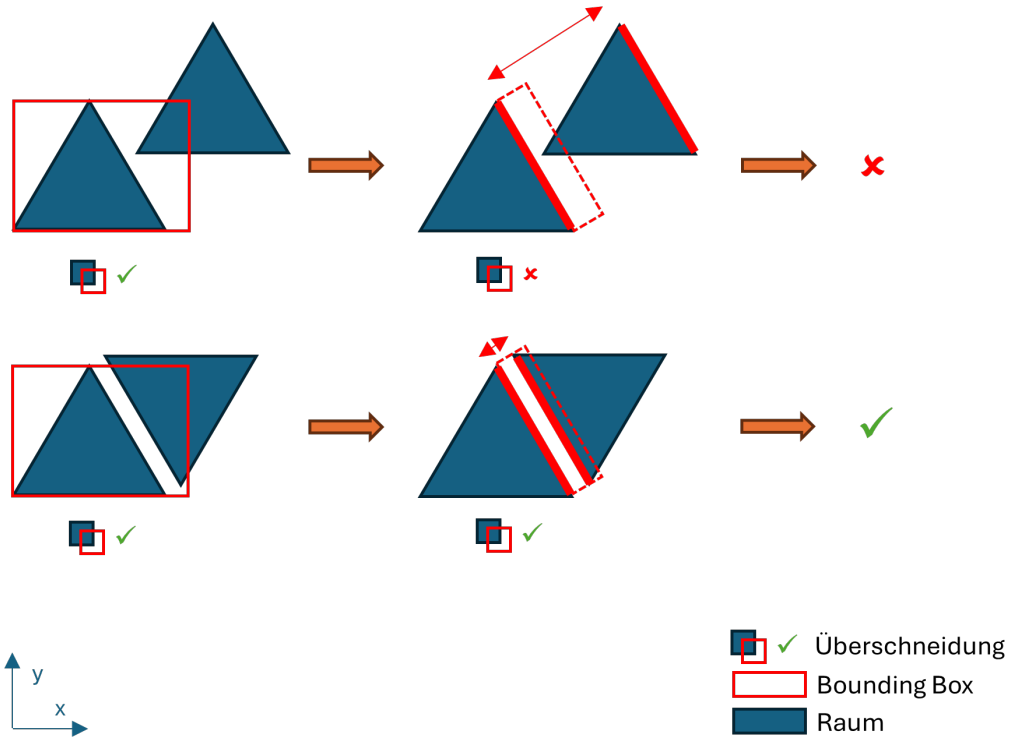



Abbildung 6.7: Prüfung der Raumnachbarschaft über die Parallelität und Nähe von Polygonkanten.

*AABB vs. erweiterte AABB*-Algorithmus um die dritte Dimension erweitert werden und diese Vorauswahl anschließend mittels je einer Einzelvergrößerung der Bounding Box nach oben und unten auf Nachbarschaften über bzw. unterhalb des Raums überprüft werden. Alternativ verzichtet man auf eine gemeinsame Vorauswahl von Lage und Höhe und prüft direkt mit einer kleineren Bounding Box-Erweiterung und hat dadurch weniger irrtümliche Funde in der Auswahl.

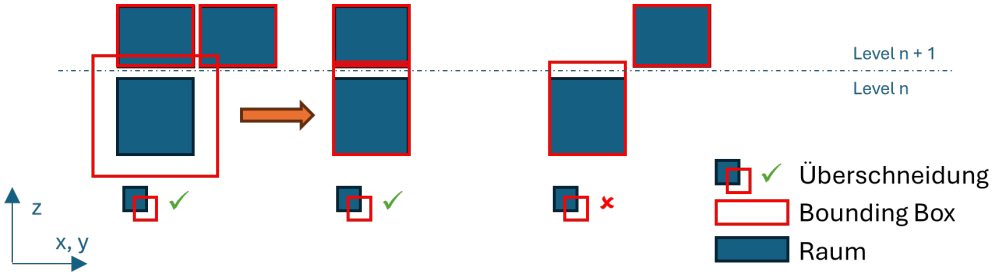



Abbildung 6.8: Darstellung der Prüfung von übereinanderliegenden Räumen.

Die obige Darstellung verdeutlicht, dass es für die vertikale Betrachtung effizienter ist, direkt mit einer nur nach oben erweiterten Bounding Box zu prüfen. Gleichzeitig darf aus dieser Darstellung nicht geschlossen werden, dass eine Prüfung mittels der Polygone entfallen kann. Für die Überschneidung in horizontaler Richtung sollte man direkt die Polygone beider Räume vergleichen. Liegt ein Punkt des einen Raums innerhalb des Polygons des anderen Raumes, so liegen beide Räume über- bzw. untereinander. Sonderfälle wie Abb. 6.4, also dass sich zwei Flächen überlagern, ohne dass ein Punkt in der Fläche des anderen liegt sind möglich. Um für jeden Raum einzeln entscheiden zu können, ob er

oberhalb oder unterhalb des Bezugsraums liegt, muss die Prüfung für jeden Raum in die jeweilige Richtung einzeln erfolgen.

Die vertikale Prüfung setzt sich also zusammen aus einer Prüfung, ob zwei Räume an eine gemeinsame Ebene angrenzen, also die Decke des einen Raums an den Boden des anderen angrenzt und dem anschließenden Kollisionstest, ob sich beide Grundriss-Polygone überlagern oder nicht. Wenn sich die Polygone überschneiden oder ineinander liegen und die Fußbodenhöhe des oberen Raums innerhalb der erweiterten AABB des unteren Raums liegt, dann grenzen die beiden Räume vertikal aneinander.

Wie auch bei der vorherigen Überlegung wird hierbei davon ausgegangen, dass alle Räume horizontale Boden- und Deckenflächen ohne Sprünge haben. Bei einer kombinierten Betrachtung von Lage und Höhe sollten alle Räume einer Ebene die gleiche Start- und Endhöhe besitzen, oder Vorkehrungen getroffen werden, um auch Höhenversätze zu berücksichtigen.

### 6.1.3 Praktische Umsetzung der beiden Überlegungen

Die Umsetzung erfolgt in *Python 3.8* (PYTHON.ORG, 2024). Als verwendete Bibliotheken sind im wesentlichen *IfcOpenShell* (IFCOPENSHELL, 2024) und *RDFLib* (RDFLIB TEAM, 2023) hervorzuheben. Wobei *RDFLib* erst bei der Übersetzung der Ergebnisse in *RDF-Triples* im Anschluss genutzt wird.

AABB und Raum-Geometrie sind durch *IfcOpenShell* für jeden Raum leicht aus dem *IFC*-File extrahierbar. Der einzig weitere nötige Input neben der *IFC*-Datei ist ein Abstandswert zur Definition der Nähe, ab der Räume als benachbart gelten, das heißt, dass z.B. die Bounding Box in der Berechnung um genau diesen Wert in jede Richtung erweitert wird. In den Tests wurde als Wert die größte Wanddicke plus 20 cm verwendet und damit sehr gute Ergebnisse erzielt. Eine Optimierung des Werts wurde nicht durchgeführt. Allerdings war festzustellen, dass die reine maximale Wanddicke nicht ausreicht und ein gewisser zusätzlicher Puffer nötig ist.

Für die horizontale Nachbarschaftsuntersuchung wird zunächst mit einem modifizierten *erweiterte AABB vs. erweiterte AABB*-Test eine Vorauswahl getroffen. Modifiziert bedeutet hier, dass nur in der X- und Y-Richtung erweitert und auf Kollisionen geprüft wird und gleichzeitig in Z-Richtung ohne Vergrößerung der Box getestet wird, ob beide Räume auf gleicher Höhe sind. Die weitere Umsetzung basiert auf der oben beschriebenen Variante 2.2. Der Algorithmus weicht von der oben beschriebenen Theorie dahingehend ab, dass im Ablauf parallel geprüft wird, ob sich die Polygone überschneiden oder ineinander liegen. Dadurch kann die Berechnungszeit im Vergleich zu einer separaten Prüfung verkürzt werden. Es wird dafür zunächst geprüft, wo die Punkte des einen Polygons relativ zum anderen liegen. Wenn ein Punkt innerhalb des anderen Polygons liegt, dann kann eine weitere Betrachtung des Nachbarschaftsfalls entfallen, da bereits klar ist, dass die Räume nicht nur nebeneinander liegen. Teile des Algorithmus basieren auf bereits etablierten Algorithmen aus den Bereichen *Kollisionstests* und *Rendering*. In beiden Bereichen ist der

Fall, dass ein Punkt genau auf der Kante oder Ecke eines anderen Polygons liegt selten genau definiert. In dynamischen Umgebungen wie in Computerspielen oder Simulationen, wofür diese Algorithmen entwickelt wurden, tritt dieser Grenzfall sehr selten auf und es ist dort meist nicht wichtig, ob ein Objekt bereits getroffen wurde oder noch nicht (HAINES, 1994). In diesen Fällen würde der benachbarte Strahl im Render-Prozess oder die Kollisionsprüfung des nächsten Bewegungsinkrements Klarheit bringen. Da die Lage der betrachteten Objekte hier aber unveränderlich ist, muss auch dieser Grenzfall extra berücksichtigt werden. Liegt ein Punkt auf einer Kante, so gilt er zunächst als *unbestimmt* und die Lage der restlichen Punkte des Polygons entscheiden über dessen Bedeutung. Liegen die anderen Eckpunkte des Raumes außerhalb des zweiten Raumes, so gilt auch der Punkt auf der Linie als *außerhalb*. Damit steht fest, dass sich die Räume nicht überschneiden. Sollten umgekehrt alle Punkte im anderen Raum liegen, so gelten auch die Grenzfälle als *innen* und damit liegen die Räume ineinander. Wenn Punkte auf beiden Seiten liegen, so überschneiden sich beide Räume teilweise und eine nähere Bestimmung des Punktes ist nicht nötig. Für den Fall, dass einer der Räume nicht konvex ist oder ein Loch hat, so müssen zusätzlich noch alle Kanten auf Überschneidungen geprüft werden oder der Test in beide Richtungen durchgeführt werden (Prüfen der Punkte von Raum *1* gegen Umriss von Raum *2* und anschließend umgekehrt).

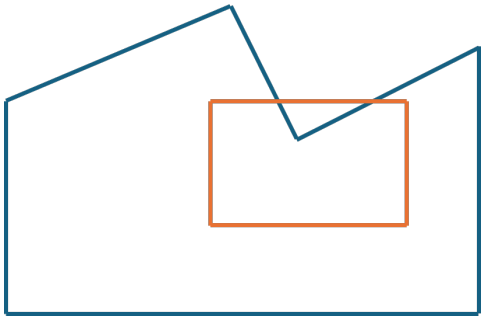


Abbildung 6.9: Die Polygone liegen nicht vollständig ineinander, obwohl alle orangenen Eckpunkte innerhalb des Blauen liegen.

Daher gestaltet sich der Algorithmus wie folgt:

1. Es wird für jeden Punkt des ersten Polygons (*P1*) bestimmt, ob dieser innerhalb, auf einer Kante, oder außerhalb des anderen Polygons (*P2*) liegt. Der eigentliche „Punkt in Polygon“-Algorithmus stammt von Eric Haines (1994) („Crossing Algorithm“) und wurde nur von *C++* nach *Python* übersetzt und in den restlichen Algorithmus integriert.

2. Für alle Kanten von *P1* wird geprüft, welchen Status die beiden Eckpunkte haben. Auch diesen Kanten wird ein Status zugeordnet:

   - Beide Punkte liegen innerhalb, oder einer innerhalb und einer auf der Kante: Kante *innerhalb*
   - Beide Punkte liegen außerhalb, oder einer außerhalb und einer auf der Kante: Kante *außerhalb*

- Ein Punkt innerhalb und einer außerhalb: Kanten schneiden sich und damit auch die Polygone. Somit überschneiden sich die Räume und der Test kann vorzeitig abgebrochen werden.
- Beide Punkte auf einer Kante: Kante *unbestimmt*

3. Da der Algorithmus die Ausnahme aus Abb. 6.4 nicht berücksichtigt, muss extra geprüft werden, ob sich Kanten der Polygone schneiden. Bei einem gefundenen Schnittpunkt wird der Test ebenfalls mit dem Status *Überschneidung* beendet. Es bleibt nun also nur noch *innerhalb* oder *außerhalb* als Status für die Polygone zueinander übrig.

4. Nachdem für alle Kanten der jeweilige Status bestimmt wurde, müssen diese für das gesamte Polygon P1 verglichen werden:

   - Liegen alle Kanten von *P1 innerhalb P2* oder sind *unbestimmt*, oder alle sind *unbestimmt*, so liegt *P1* in *P2*, insofern *P1* auch nicht über den Höhenbereich von *P2* hinausragt. Sonst handelt es sich wieder um eine *Überschneidung* und der Algorithmus wird beendet.
   - Liegen alle Kanten *außerhalb* oder sind *unbestimmt*, dann liegen die Polygone entweder nahe nebeneinander oder *P2* liegt in *P1*. Durch eine Überprüfung der Punktstatus von *P2* gegenüber *P1* lässt sich auch dieser Fall unterscheiden. Wenn alle Punkte von *P2* in *P1* liegen und die Höhen wie im obigen Punkt passen, dann liegt *P2* in *P1* (oder es liegt ggf. eine vertikale *Überschneidung* vor), sonst liegen sie nebeneinander, da alle anderen Fälle schon ausgeschlossen wurden.

   Sollte einer der Punkte zutreffen, wird der Algorithmus mit dem Status *P1 liegt in P2* oder *P2 liegt in P1* beendet. Es bleiben nur noch nahe liegende Räume wie zum Ende von Schritt *2* der Überlegung übrig.

5. Wenn also die Fälle *Überschneidung* und *innerhalb* ausgeschlossen wurden, folgt ein Vergleich aller Kanten untereinander auf Parallelität (Kreuzprodukt der Kanten = 0). Sobald ein paralleles Kantenpaar gefunden wurde, wird mittels *Ray–line segment intersection*-Test überprüft, ob beide Kanten auch mindestens teilweise auf gleicher Höhe liegen. Der Test basiert auf dem *ray-triangel intersection*-Algorithmus von Möller und Trumbore (1997), welcher auf eine statt drei Strecken reduziert wurde.

Am Ende dieses Tests steht nun für jede Raumpaarung fest, ob sie tatsächlich *nebeneinander* liegen, oder nur nahe zusammen, aber keine tatsächliche räumliche Beziehung haben. Je nachdem, wie streng die Parallelitätsprüfung eingestellt ist, kann es sein, dass es bei, durch Polygonzüge angenäherten, gekrümmten Wänden zu Problemen kommt, da die Polygonzüge der gekrümmten Kanten nicht zwingend parallel laufen. Auch zählen für diesen Algorithmus zwei Räume noch als benachbart, wenn sich lediglich zwei Endpunkte überlagern (vgl. untere Abbildung). Eine zusätzliche Implementierung einer Mindestüberlappungslänge könnte dieses Problem in Zukunft lösen. Diese Länge könnte

prozentual sein, oder ein fixer Überschneidungsbereich, der mindestens notwendig ist, damit die Raumnachbarschaft auch praktisch genutzt werden kann. Damit würde zu jedem Eckpunkt ein, auf der Kante um diesen Abstand nach innen versetzter „Mindestüberlagerungspunkt“ gehören und erst wenn mindestens ein solches Punktepaar an den Strahlen von beiden Punkten aus den anderen Raum detektiert, dann soll von einer Nachbarschaft ausgegangen werden.


Abbildung 6.10: Grenzfall: Räume werden noch als Nachbarn erkannt, obwohl sie sich nur in einem Punkt überlappen

Für den vertikalen Test wird erst geprüft, ob Räume vertikal aneinander grenzen und nicht auf gleicher Höhe sind. Dann folgt ein *AABB vs. AABB*-Test in der Lage. Nachdem nun alle übrig gebliebenen Kombinationen nahe zusammen und potenziell übereinander liegen und sich dieses Mal zwingend überschneiden müssen, werden für jedes Raumpaar „Punkt in Polygon“-Tests (HAINES, 1994) durchgeführt. Dabei werden direkt beide Polygone gegeneinander getestet, da nun nicht unterschieden werden muss, ob ein Polygon innerhalb des anderen liegt oder umgekehrt. Sobald mindestens ein Punkt innerhalb des anderen Polygons liegt, liegen die entsprechenden Räume übereinander. Dadurch ergibt sich nur noch der Spezialfall wie in Abb. 6.4, wo kein Eckpunkt im anderen Polygon liegt, aber die Räume sich doch überlagern. Dafür müssen auch hier wieder zusätzlich die Kanten auf Schnittpunkte untereinander geprüft werden. Da die Polygone auch ohne Schnittpunkt ineinander liegen können, kann dieser Test den ersten nicht ersetzen. Sobald damit gesichert ist, dass die beiden getesteten Räume übereinander liegen, wird noch einmal genauer überprüft, ob der erste angegebene Raum über dem zweiten liegt, oder umgekehrt. Auch in diesem Fall könnte man über eine „Mindestüberschneidungsfläche“ ähnlich der Überschneidungslänge in der Horizontalen nachdenken.

Bei dieser Implementierung treten zwei größere Probleme auf:

Das erste Problem entsteht, wenn ein Raum aus mehreren Polygonen besteht, also ein Loch hat, da dann diese Polygone beim Auslesen durch *IfcOpenShell* aneinander gehängt werden. Da der Prozess nicht darauf ausgelegt ist, zwei Polygone pro Raum zu betrachten, werden die Punkte als ein einziges Polygon interpretiert. Dadurch wird jedoch mindestens eine Außenkante „zerstört“. Dies ist anhand eines Beispiels in der folgenden Abbildung illustriert. Das blaue Polygon hat ein quadratisches Loch und durch die Verkettung beider Polygone wird genau die Kante gestört, welche an den orangenen Raum angrenzen sollte. Dadurch wird die Nachbarschaftsbeziehung der beiden Räume nicht korrekt erkannt.

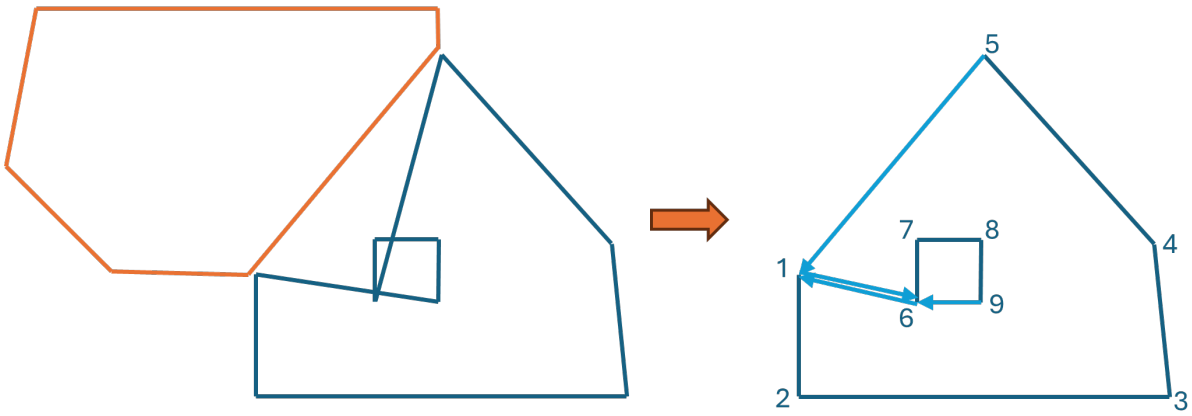



Abbildung 6.11: Ergebnis nach der Verarbeitung von Polygonen mit Löchern und wie man diese reparieren kann.

Um das Polygon äußerlich zu „reparieren“, wird in der Punktliste vor dem ersten Punkt des inneren Polygons noch einmal der Startpunkt des äußeren Umrisses eingefügt. Damit ist das äußere Polygon bereits wieder geschlossen. Dasselbe wird auch mit dem inneren Polygon (oder auch mehreren) wiederholt. Hierfür wird der erste Punkt des inneren Polygons an das Ende der Punktkette für dieses Loch kopiert. Damit es wieder eine geschlossene Fläche gibt, wird anschließend noch einmal der erste Punkt des äußeren Polygons ergänzt. Damit ist die Fläche theoretisch durch ein einziges Polygon beschrieben. Das Loch ist durch einen „unendlich schmalen“ Spalt mit dem „Außenbereich“ verbunden und es existiert dadurch theoretisch keine abgeschlossene Fläche mehr im Loch. Der Flächeninhalt kann dann über gängige Formeln berechnet werden und ändert sich nicht. Die fertige Lösung ist in der obigen Abbildung rechts zu sehen. Es wurde dort also aus der ausgelesenen Punktreihenfolge (1, 2, 3, 4, 5, 6, 7, 8, 9) die „reparierte“ Folge (1, 2, 3, 4, 5, **1**, 6, 7, 8, 9, **6**, **1**,). Die entstandene „Doppelkante“ wird anschließend bei den Schnittberechnungen der Kanten herausgefiltert, um Fehldetektionen zu vermeiden. Für diese Lösung dient die *Half-Edge*-Darstellung bzw. *Boundary Representation* Methode als Grundlage (vgl. OHORI, 2021, BERG et al., 2008).

Die Punkte werden auch von *IfcOpenShell* meistens in der korrekten Reihenfolgen ausgelesen und gespeichert, wie es bei den genannten Prinzipien üblich ist. Für die Raumflächen werden die Punkte gegen den Uhrzeigersinn angegeben und für Löcher im Uhrzeigersinn. Dass die Reihenfolge gelegentlich nicht eingehalten wird, führt allerdings zum zweiten Problem. Da es sich bei den Räumen um dreidimensionale Körper handelt, werden neben den Punkten im Grundriss auch die Eckpunkte auf Deckenhöhe oder dazwischen ausgegeben. Speziell diese Reihenfolge variiert gelegentlich, weshalb vor allen anderen Berechnungen ein Sortierschritt eingeführt werden musste, der sicherstellt, dass alle Punktlisten richtig sortiert sind. Für die Sortierung werden nun nicht nur die Punkte ausgefiltert, die nicht auf der untersten Ebene liegen (Höhe > 0), sondern es wird auch die Liste der Kanten betrachtet. Bringt man die Kanten der Punkte, die zuvor ausgewählt wurden, dann in eine Reihenfolge, sodass der Endpunkt einer Kante immer dem Anfangspunkt der nachfolgenden entspricht, dann kann man daraus die korrekte Sortierung der Punkte ermitteln.

## 6.2 Erreichbarkeit von Räumen

Wie man erkennt, welche Räume eine Tür oder ein Durchgang verbindet, wurde bereits zuvor beschrieben. *IfcOpenShell* (IFCOPENSHELL, 2024) vereinfacht die Untersuchung durch interne Berechnungen. Jeder *IfcDoor* wird beim einlesen einer IFC-Datei mit *IfcOpenShell* automatisch ein Attribut zugewiesen (*ProvidesBoundary*), welches Information darüber enthält, welche Räume dieses Objekt begrenzt. Bei einer Tür sind das die Räume, die durch diese Tür verbunden sind. Durch das Auslesen aller Türen mit zwei zugewiesenen Räumen erhält man alle Paare für die Relation *rao:hasAccessTo*. Es wird vereinfachend angenommen, dass alle Räume über Elemente des Typs *IfcDoor* verbunden werden.

## 6.3 Umwandlung von Property Sets

In IFC werden alle Informationen in *Property Sets* thematisch gebündelt (BUILDINGSMART INTERNATIONAL, 2019). Von IFC selbst wird das Set *Qto_SpaceBaseQuantities* ausgelesen. Darin befinden sich die geometrischen Eigenschaften wie Fläche, Höhe und Volumen, welche später unter *rao:GeometricProperty* gesammelt werden. Zusätzlich dazu wird ein benutzerdefiniertes *Property Set* ausgelesen. Dieses muss zuvor im Autorenprogramm definiert werden. Darin werden alle Informationen zu erlaubten, geforderten und bereitgestellten Ressourcen gesammelt, sowie die Personengruppen, die Raumausstattung und sonstige, nicht standardmäßig in *IFC* definierte, Werte.

Um bei den erlaubten Durchleitungen nicht alle Ressourcen einzeln jedem Raum zuordnen zu müssen, wurden die Keywords *All* und *None*, sowie ein Minuszeichen als Präfix für *nicht* eingeführt. Sollten also in einem Raum alle Ressourcen erlaubt sein, dann muss als Eigenschaft lediglich „All“ übergeben werden anstelle einer einzelnen Aufzählung der Ressourcen, was bei nachträglichen Ergänzungen fehleranfällig ist. Es waren keine klaren Vorteile zwischen einer Variante, in der nur die verbotenen, oder einer, in der nur die erlaubten Ressourcen aufgezählt werden, erkennbar. Es wurde sich für eine Aufzählung der Erlaubten entschieden. Sofern die Liste der verbotenen Ressourcen kürzer ist, wird ein Minuszeichen vor dem Ressourcennamen verwendet. Dieses wird im Folgenden dann durch den Algorithmus so interpretiert, dass alles außer diese markierten Ressourcen erlaubt ist. Es gibt hier allerdings keine Kontrollstruktur auf der Eingabeseite, welche prüft, dass dort auch sinnvolle und widerspruchsfreie Werte eingegeben wurden.

## 6.4 Erstellung des Graphs

Für die Erstellung des Graphen nach den Regeln der *RAO* wird das Paket *rdflib* (2023) verwendet. Dafür werden alle zuvor gesammelten Informationen in einen Graph mit einer zuvor festgelegten *IRI* überführt. Dieser Algorithmus setzt voraus, dass es die Klassen *Höhe*, *Fläche*, und *Volumen* als Unterklassen von *rao:GeometricProperty* gibt und sonst keine weiteren Zwischenstrukturen in den Anforderungen definiert werden.

Alle anderen Eigenschaften, Ressourcen, etc. werden dann direkt den entsprechenden Klassen zugeordnet. Für die betrachteten Beispiele ist diese Übersetzung ausreichend. Differenziertere Modellschemata wurden nicht berücksichtigt, könnten aber Teil künftiger Untersuchungen sein.

## 6.5 Zusammenführung aller Prozesse

Um die definierten Abfragen und die Funktionsfähigkeit der Algorithmen zu testen, werden nachfolgend zwei Beispiele gezeigt. Die Gebäude wurden dafür in *Revit 2024* (AUTODESK, 2024) modelliert und mit Eigenschaften versehen. Die Modelle werden anschließend in das IFC-Format exportiert (*IFC 4 Reference View*). Anschließend wird die *IFC*-Datei einmal mit dem *LBD-Converter* (BONDUEL et al., 2018) und dem selbst entwickelten Script ausgewertet. Damit erhält man zwei Graph-Dateien im *.ttl*-Format. Eine Verknüpfung war explizit nicht gewollt, da die meisten, bisher gezeigten, Abfragen ausschließlich letzteren Graph benötigen und somit beide Dateien getrennt betrachtet werden können, was die Anzahl an Tripeln in der Datenbank reduziert und damit die Effizienz erhöht. Der Graph aus dem eigenen Converter enthält alle *RAO*-Relationen und die *BOT*-Raumbeziehungen und ist meistens für die Beantwortung der zuvor aufgestellten Abfragen ausreichend. Im Ergebnis des *LBD-Converters* sind die Informationen über Geschosse enthalten, sowie alle anderen Relationen von *BOT* und *PROPS*, welche aber nicht für diesen Zweck benötigt werden. Dieser Graph wird nur hinzugezogen, wenn es für die jeweilige Abfrage nötig ist, da durch die Verknüpfung alle Räume zweimal repräsentiert werden und dies trotz funktionierender Verknüpfung in vielen Abfragen zu Problemen führt. Es tauchen dann nämlich in den Ergebnissen alle Räume jeweils als Instanz aus der *IFC2LBD*-Datei und aus der eigenen Datei auf. Damit die Graphen verknüpft werden können muss im *LBD-Converter* eingestellt sein, dass auch die *ifcOWL*-Datei mit ausgegeben und verlinkt wird. Die *IRIs* in der LBD-Datei sind nämlich mit neuen GUIDs versehen, die nicht den *IFC-GUIDs* entsprechen, daher nicht reproduzierbar und damit nicht zum verlinken geeignet sind. Wird allerdings die *ifcOWL*-Datei verknüpft, dann wird über *owl:sameAs* die *IRI* des Objekts aus dieser Datei verlinkt. Diese *IFC*-Bezeichnung ließ sich leicht nachbauen und für das selbst entwickelte Script verwenden. Zudem muss kontrolliert werden, ob alle Präfix-*URLs* korrekt sind. In beiden Graphen werden die Individuals unter dem Präfix *inst:* deklariert. Die *URL* dafür muss deshalb in beiden Fällen exakt gleich sein. Wenn die *ifcOWL*-Datei bereits vor dem Konvertieren vorhanden ist, dann verändert sich zwar die Präfix-*URL* in der LBD-Datei, aber die Verlinkungen der *owl:sameAs*-Beziehungen sind korrekt und ohne falschen absoluten Dateipfad in der IRI. Als *Triplestore* wird *GraphDB* (ONTOTEXT, 2024b) verwendet. Damit werden auch die *SPARQL*-Abfragen durchgeführt.

# Kapitel 7

# Anwendungsfälle

## 7.1 Beispiel 1: Bürogebäude

### 7.1.1 Beschreibung des Gebäudes

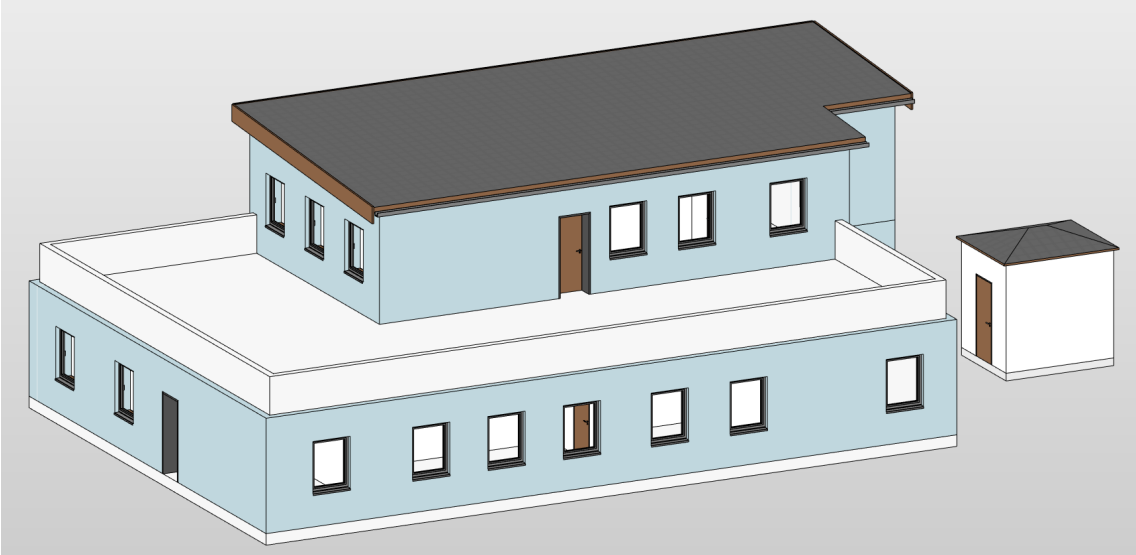


Abbildung 7.1: Ansicht des fiktiven Bürogebäudes mit angrenzender Gartenhütte in *Revit*

Das Beispielobjekt besteht aus einem zweistöckigen Bürogebäude, sowie einer angrenzenden Gartenhütte. Die Grundrisse hierzu befinden sich im Anhang A.

Das Haus besteht im Erdgeschoss aus mehreren Büroräumen, sowie der Haustechnik (Strom, Wasser, Server) und einem Pausenraum mit Küche. Das obere Geschoss hat eine kleinere Grundfläche und ist über ein Treppenhaus mit dem Erdgeschoss verbunden. Hier befinden sich weitere Büroräume. Die übrige Fläche wird als Dachterasse genutzt. Beide Stockwerke verfügen zudem über je einen Toilettenraum. Die Gartenhütte ist nicht physisch mit dem Hauptgebäude verbunden.

Da das Modell eine frühe Leistungsphase repräsentieren soll, sind keine Einrichtungen modelliert und es werden als interne Anforderungen nur geometrische Anforderungen geprüft. Damit lassen sich die Abfragen vollumfänglich testen und eine Berücksichtigung von z.B. verbauten Heizungen oder anderen Elementen bei der Prüfung würde lediglich einen höheren Konvertierungsaufwand bedeuten, welcher gegebenenfalls auch projektspezifisch konfiguriert werden muss.

### 7.1.2 Fiktiver Regelsatz

Der folgende Regelsatz für dieses Gebäude stellt die Anforderungen des Besitzers dar und ist nicht darauf ausgelegt möglichst realistisch zu sein, sondern soll nur einen möglichen Prüfungsprozess zeigen. Als zu betrachtende Ressourcen (*rao:Resource*), werden für dieses Projekt *Wasser*, *Strom* und *Netzwerk* für die Gebäudetechnik, sowie *Erholung* und *Toilette* als Bedürfnisse von Mitarbeitern definiert. Bei den Personengruppen (*rao:Group*) wird zwischen *Mitarbeitern* und *Besuchern* unterschieden, wobei Mitarbeiter auch in alle Besucherräume dürfen. Die internen Eigenschaften der Räume sind in diesem Fall rein geometrisch (*rao:GeometricProperty*): *Höhe*, *Fläche* und *Volumen*. Die Raumtypen (*rao:Function*) werden in der folgenden Tabelle zusammen mit ihren Anforderungen aufgeführt.

Tabelle 7.1: Anforderungsliste der einzelnen Räume

| **Raumtyp** (*Function*) | **benötigt** (*needs*) | **liefert** (*provides*) | **verbietet** (invers zu *allowsPassingOf*) | **Zugang für** (*accessibleFor*) | **liegt neben:** (*isNextTo*) |
|---|---|---|---|---|---|
| Büro | Erholung<br>Netzwerk<br>Strom | - | Wasser | Mitarbeiter | - |
| Chefbüro | gleich wie Büro | | | | Sekretariat |
| Flur | Strom | - | - | Besucher | - |
| Küche | Strom<br>Wasser | - | Netzwerk | Mitarbeiter | Pause |
| Lager | Strom | - | - | Mitarbeiter | - |
| Pause | Netzwerk<br>Strom | Erholung | - | Mitarbeiter | Küche |
| Sekretariat | gleich wie Büro | | | Besucher | Chefbüro |
| Server | Netzwerk<br>Strom | Netzwerk | Wasser | Mitarbeiter | - |
| Stromversorgung | Strom | Strom | Wasser | Mitarbeiter | - |
| Treppenhaus | Strom | - | - | Besucher | Flur |
| Wasserversorgung | Strom<br>Wasser | Wasser | Strom<br>Netzwerk | Mitarbeiter | - |
| WC | Strom<br>Wasser | Toilette | Netzwerk | Besucher | - |

In dieser Liste fehlen allerdings aufgrund der Menge die geometrischen Anforderungen. Für die einzelnen Raumtypen sind Richtwerte für Raumhöhe, Grundfläche und Volumen festgelegt. Diese Maße können entweder absolut angegeben sein, oder über einen

minimalen und maximalen Wert. Sofern relevant, werden diese Regeln in den Abfragen ersichtlich.

### 7.1.3 Auswertung des Regelsatzes

Die Analyse des Gebäudes erfolgt über die zuvor definierten Abfragen und einigen zusätzlichen Abwandlungen davon, wenn es sich anbietet, um weitere Fehler im Modell oder andere interessante Aspekte zu zeigen. Es wird also geprüft:

1. die Modellvollständigkeit, dass jeder Raum vorhanden ist,
2. die Plausibilität, ob alle Räume erreicht werden können,
3. ob die geometrischen Anforderungen erfüllt werden,
4. ob alle Raumbedürfnisse erfüllt werden können,
5. ob die Räume nebeneinander liegen, die es sollen,
6. ob die Anzahl an speziellen Räumen korrekt ist.

Zuvor wurde noch per Hand geprüft, ob tatsächlich alle Raumbeziehungen korrekt erkannt wurden (*bot:adjacentZone* und Unterklassen). Die Prüfung hat ergeben, dass alle 46 horizontalen und alle 23 vertikalen Nachbarschaften, sowie korrekterweise keine Überschneidungen und Überlagerungen erkannt wurden. Allerdings ist bei der Prüfung aufgefallen, dass bei der Tür zwischen Treppenhaus und oberem Flur in *BIMvision* (DATACOMP, 2024) keine Verbindung erkannt wurde, was darauf schließen lässt, dass auch im Graph diese Türe nicht korrekt als Durchgang zwischen den Räumen erkannt wurde. Der Fehler wurde in diesem Zuge händisch im Graph ausgebessert. Das Problem entsteht hier bereits im *IFC*-Export bzw. bei der Modellierung in *Revit* und konnte bis zum Abschluss dieser Arbeit zwar reproduziert aber nicht gelöst werden.

Es werden nun die Abfragen aus Kapitel 5 auf das Modell angewandt. Dies stellt vereinfacht die auftraggeberseitige Prüfung eines übergebenen Planungsstands dar.

#### 1. Modellvollständigkeit

Die erste Abfrage, ob alle in den Anforderungen definierten Räume modelliert wurden, ergibt, dass dort in der Liste auch ein Erste-Hilfe-Raum vorgesehen war, welcher aber nicht modelliert wurde. In der Realität hätte man, bereits bevor der erste Mitarbeiter die Pläne gelesen hätte, einen Grund gefunden, den Planungsstand abzulehnen und nachbessern zu lassen.

### 2. Plausibilität und Erreichbarkeit

Die Erreichbarkeitsanalyse aus **Abfrage 2** zeigt, dass ein Raum nicht erreicht werden kann (Ergebnis: „1“). Dabei handelt es sich um die Gartenhütte, welche nicht direkt mit dem Bürogebäude verbunden ist. Das Problem kann daher ignoriert werden, oder durch die Modellierung des Außenbereichs als Raum gelost werden.

Wenn nun aber geprüft wird, ob alle für Besucher freigegebenen Räume erreichbar sind, so steigt die Zahl der unverbundenen Räume auf zwei. Für die Prüfung wurde die **Abfrage 2** so modifiziert, dass nur Räume, die Besucher erlauben, berücksichtigt werden. Der Grund für den weiteren Raum neben der Gartenhütte, ist die Dachterrasse, welche für Besucher nicht erreichbar ist, da der Zugang durch ein Büro führt, welches nur für Mitarbeiter freigegeben ist.

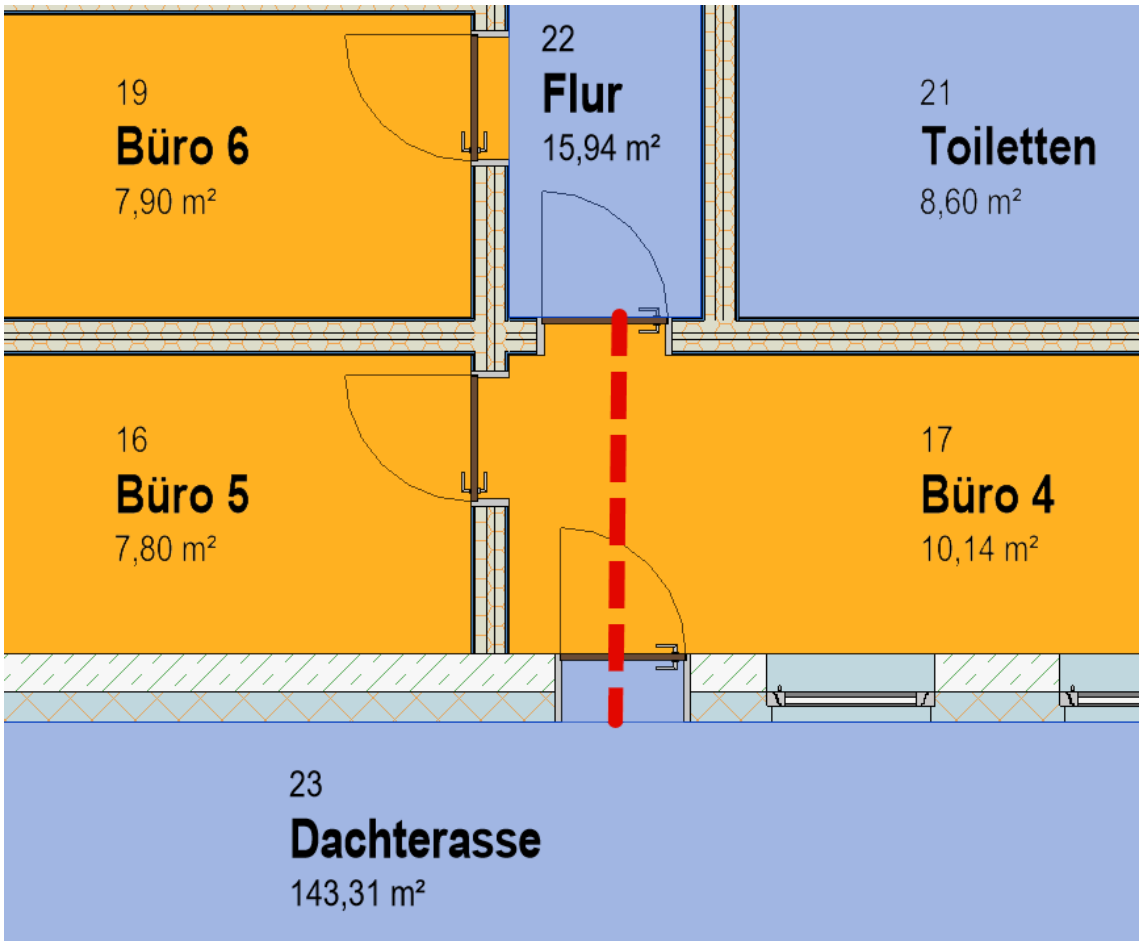



Abbildung 7.2: Auschnitt aus dem OG. Der Weg zwischen zwei Räumen mit Besucherfreigabe (blau) wird durch einen Raum ohne Freigabe (nur für Mitarbeiter) durchschnitten.

### 3. Geometrische Regelprüfung

Anschließend werden auch die geometrischen Mindestanforderungen geprüft, dafür wurden im Regelwerk fiktive Maße für Fläche, Höhe und Volumen jedes Raumtyps festgelegt und in der **Abfrage 3** mit den tatsächlichen Raummaßen im Modell verglichen. Die nach-

folgende Tabelle zeigt die Auswertung dieser *SPARQL*-Abfrage. Es werden dabei als Information die Raum-*IRI*, der Raumname, die geprüfte Kategorie (*Eigenschaft*) und der Raumtyp (*Funktion*) von dem die Anforderung kommt, ausgelesen. Danach folgt die Spalte mit dem ausgelesenen reellen Wert und dann die Paare aus Anforderungswert und der Ausgabe, ob gegen diesen Wert verstoßen wurde, jeweils für konstante Werte, sowie für Minimal- und Maximalwerte. Je nachdem von welchem Typ die Anforderung war, erscheint die Auswertung in der entsprechenden Spalte.

In der Tabelle tauchen einige Verstöße auf. Während die Überschreitungen in diesem Beispiel gewollt sind, würden alle diese Probleme den Fachplanern z.B. durch einen BIM-Koordinator wieder zum nachbessern zurückgeschickt. Als erstes fallen zwei Räume des Typs *inst:Lager* auf, welche eine zu hohe Decke besitzen. In den Anforderungen wurde für *Lager* eine Deckenhöhe von 2,5 m ohne Toleranz festgelegt. Mit 2,8 m und 3,0 m liegen die Raumhöhe der Gartenhütte und des Abstellraums beide über dem Grenzwert.

Für Erholungsflächen wurde die Funktion *Pause* mit Anforderungen an die Grundfläche des Raumes definiert. Neben dem Pausenraum im EG wurde auch die Dachterrasse aufgrund ihrer Erholungsfunktion als Pausenraum deklariert. Es wird deshalb in der Tabelle festgehalten, dass die Dachterrasse mit 143,31 m² zwar nicht gegen die Mindestanforderung von 20 m² als Pausenraum verstößt, aber deutlich über der Maximalbegrenzung von 50 m² liegt. Die restlichen Verstöße sind analog zu lesen und zeigen dass die jeweiligen Räume entweder ein zu geringes Raumvolumen oder eine zu kleine Grundfläche haben.

Tabelle 7.2: Ergebnis der Auswertung der geometrischen Mindestanforderungen mit SPARQL (**Abfrage 3**)

| Raum | Raumname | Eigenschaft | Funktion | value | reqVal | match Violated | reqMin | min Violated | reqMax | max Violated |
|---|---|---|---|---|---|---|---|---|---|---|
| inst:IfcSpace_2579 | Gartenhütte | inst:Height | inst:Lager | 3,0 | 2,5 | true | | | | |
| inst:IfcSpace_2413 | Dachterasse | inst:Area | inst:Pause | 143,31 | | | 20 | false | 50 | true |
| inst:IfcSpace_693 | Elektrik | inst:Volume | inst:Strom-versorgung | 19,03 | | | 20 | true | | |
| inst:IfcSpace_1384 | Flur | inst:Area | inst:Flur | 7,7 | | | 10 | true | 50 | false |
| inst:IfcSpace_1097 | Abstellraum | inst:Height | inst:Lager | 2,8 | 2,5 | true | | | | |
| inst:IfcSpace_1506 | Treppenhaus | inst:Area | inst:Treppen-haus | 8,8 | | | 20 | true | 30 | false |

Tabelle 7.3: Ausgabe der **Abfrage 4.1** bezüglich der Anzahl an Räumen je Ressource, zwischen denen keine Verbindung hergestellt werden kann.

| res | count |
|---|---|
| inst:Strom | 1 |
| inst:Wasser | 1 |
| inst:Netzwerk | 1 |
| inst:Erholung | 0 |
| inst:Toilette | 0 |

Tabelle 7.4: Ausgabe der **Abfrage 4.2**. Aufgelistet werden alle nicht erfüllten Raum-Ressourcen-Kombinationen.

| dst | res |
|---|---|
| inst:IfcSpace_2579 | inst:Strom |
| inst:IfcSpace_2579 | inst:Wasser |
| inst:IfcSpace_2579 | inst:Netzwerk |

#### 4. Raumbedürfnisse

Mit den **Abfragen 4.1** und **4.2** werden die eigentlichen Raumbedürfnisse geprüft. Hierbei wird von der Unterscheidung zwischen materiellen Ressourcen (Gebäudetechnik) und abstrakten Ressourcen (z.B. Bedürfnisse von Raumnutzern) in der Ontologie profitiert. Für die Ressourcen werden in den genannten Abfragen Pfade über die reine Nachbarschaft von Räumen, welche die Ressource erlauben, gebildet. Aber für die abstrakten Ressourcen *inst:Toilette* und *inst:Erholung*, welche ein jedes Büro von seinen Nutzern „erbt“, müsste die Überprüfung über die Zugänglichkeit wie **Abfrage 2** erfolgen. Eine Person kann bekanntlich nicht durch Wände gehen. Beide Bedürfnisse tauchen im Ergebnis der **Abfrage 4.1** (Tabelle 7.3) als vollständig erfüllt auf („0“ Pfade nicht möglich). Da aus dem Ergebnis von **Abfrage 2** bekannt ist, dass außer der Gartenhütte als Lager jeder Raum erreichbar ist, kann davon ausgegangen werden, dass dieses Ergebnis korrekt ist, auch wenn es offiziell falsch bestimmt wurde. Die Ressourcen aus dem Bereich TGA stehen jeweils mit einem nicht gefundenen Pfad in der Liste. In der präziseren Variante, nämlich **Abfrage 4.2** (Tabelle 7.4), wird ersichtlich, dass alle nicht möglichen Pfade wieder auf die externe Gartenhütte zurückzuführen sind. Dass eine Gartenhütte einen Netzwerkanschluss benötigt wäre ein Indiz, dass ein Eingabe- oder Planungsfehler vorliegt, welcher nun in der Ergebnistabelle klarer ersichtlich ist, als in der Liste der Eigenschaften des Raumes im Modell.

#### 5. Raumnachbarschaftsprüfung

Die geforderte Nachbarschaft von *Chefbüro* und *Sekretariat* wurde eingehalten und damit ist die Rückgabe der **Abfrage 5** für dieses Modell leer.

#### 6. Raumanzahl

Zur Demonstration der Prüfung von Raummengen in Abhängigkeit verschiedener Parameter, wurde die **Abfrage 6** bereits passend hierfür formuliert, da es prinzipiell unzählige Möglichkeiten für Formeln und Bedingungen gibt. Das unten gezeigte Ergebnis besagt, dass pro 100 m² Bürofläche (*inst:Büro*) genügend Toilettenräume vorhanden sind. Dass in dem zweigeschossigen Gebäude mit einer Toilette im Erd- und Obergeschoss die Bedingung „Eine Toilette je Etage“ nicht erfüllt wird, liegt daran, dass auch die Dachebene als Etage mit gezählt wird. Hierfür kann man entweder die Abfrage verfeinern, oder die Dachebene entfernen. Bei komplexeren Ebenenstrukturen, müsste noch ein Konzept entwickelt werden, um die Liste aller Ebenen zu filtern, z.B. über eine bestimmte Eigenschaft, die vergeben wird. Diese Abfrage ist die einzige, bei der der Graph aus dem LBD-Converter hinzugezogen werden muss, da nur dort Informationen zu den Geschossen enthalten sind.

Tabelle 7.5: Ausgabe der Abfrage 5: Einhaltung der Regeln zur Raumanzahl.

| Typ | Ergebnis |
|---|---|
| „Eine Toilette je 100 m² Bürofläche“ | true |
| „Eine Toilette je Geschoss“ | false |

### 7.1.4 Optimierung

In diesem Optimierungs-Beispiel soll untersucht werden, ob es einen Vorteil bezüglich der Kabelwege bringt, den Stromversorgungsraum mit dem Lager zu tauschen, da dieses zentraler liegt. Die präsentierte Variante führt zu durchschnittlichen Pfadlängen von 2,5 Räumen, welche auch die Anzahl an Durchbrüchen widerspiegelt. Tauscht man eben diese zwei Räume, dann ergibt sich eine durchschnittliche Pfadlänge von 1,9 Räumen. Da in diesem Beispiel die meisten Räume eher kompakt und nicht langgezogen oder umschließend sind, kann daraus geschlossen werden, dass sich auch die Kabellängen dadurch verkürzen werden. Das Volumen des neuen Raums ist allerdings immer noch kleiner als die geforderten 20 m³, da beide Räume fast gleich groß sind (vgl. Tabelle 7.2).

### 7.1.5 Zusammenfassung

Dieses Beispiel zeigt einen Ausschnitt der Möglichkeiten, welche die *Room Analysis Ontology* bietet, bereits in frühen Leistungsphasen Aussagen zur Anordnung der Räume zu treffen. Die Ergebnisse der Abfragen waren stets korrekt bezüglich der Anforderungen. Damit konnte gezeigt werden, dass die Ontologie und die dazu entwickelten Abfragen funktionieren.

## 7.2 Beispiel 2: Betriebsräume eines U-Bahnhofs

### 7.2.1 Ausgangslage

Mit dem Baubeginn 2022 wird nun nach 12 Jahren Pause wieder an einer neuen U-Bahnstrecke in München gebaut. Die U-Bahnhöfe in München bestehen zu einem großen Teil aus Betriebsräumen für den Bahnhof, die Tunnel und gelegentlich auch für ganze Streckenabschnitte. Die Betriebsräume müssen dabei auf beengtem Raum zwischen den Gleisen an den Bahnhofsköpfen und darüber in den Sperrengeschossen angeordnet werden. Dabei gelten einerseits geometrische Herausforderungen, wie dass große technische Bauteile, wie Transformatoren, in das Bauwerk eingebracht werden müssen und auch später wieder ausgebaut bzw. getauscht werden können müssen. Zudem benötigen viele Räume oder Raumgruppen eigene Luftschächte zur Oberfläche und Versorgungskreisläufe (z.B. getrennte Stromversorgung oder Belüftung). Neben diesen und weiteren konstruktiven Randbedingungen ergeben sich sehr viele Raumbedürfnisse innerhalb eines Bahnhofs. Diese werden zusammen mit weiteren Anforderungen an Räume und

allgemein den Innenausbau im *Richtlinienkatalog U-Bahn (RLK)* (BAU-J & SWM, 2023) festgehalten.

### 7.2.2 Richtlinienkatalog

Der RLK wird gemeinsam vom Baureferat und den SWM aufgestellt und stetig fortgeschrieben. In diesem Katalog werden die Anforderungen und Konventionen „für die Ausrüstung und die Gestaltung der U-Bahnhöfe, ihrer Betriebsräume und Einrichtungen“ (BAU-J & SWM, 2023) bei Neuplanungen definiert. Die aufgestellten Regeln sind allerdings nicht bindend, wie bei einer Norm, sondern dienen als Planungshilfe und müssen nicht vollständig eingehalten werden. Die Anforderungen werden für frühe Leistungsphasen und als Startpunkt für detailliertere Berechnungen genutzt. Wenn eine Berechnung zeigt, dass z.B. eine Flächenanforderung unterschritten werden kann und dabei trotzdem alle Bauteile sicher und nach allen Regeln der Technik in einen Raum passen, dann darf die Fläche an dieser Stelle optimiert werden. Es gibt aber auch Regelungen, welche unveränderlich sind, da sie durch andere Normen oder Verordnungen definiert werden, oder weil sie nicht optimiert oder anderweitig verändert werden können. Beispiele sind die zwingende Einhaltung des Lichtraumprofils (RLK I.2.3) und dass es keine „toten“ Räume geben darf (RLK I.2.5), also dass jeder Bereich im Bahnhof zugänglich sein muss. Letztere Anforderung spiegelt sich auch in der zweiten *Competency Question* wieder und kann durch die *RAO* und die dafür passende Abfrage ausgewertet werden.

Neben allgemeinen Anforderungen für den gesamten Bahnhof oder einzelne Bereiche, gibt es auch gesonderte Beschreibungen für jeden Raumtyp. Diese sind in der Regel folgendermaßen gegliedert:

- **Größe:** Hier werden Mindestgrößen für z.B. Grundfläche, Raumhöhe und Raumvolumen definiert. Gelegentlich gibt es neben einem Minimalwert auch einen Optimalwert, oder getrennte Maße für große und kleine Bahnhöfe. Diese Anforderungen können in der Ontologie größtenteils über die *rao:GeometricProperties* abgebildet werden. Es gibt aber z.B. auch Forderungen nach Mindestlängen von Wandabschnitten, welche mit der zuvor präsentierten Umsetzung nicht erfasst werden können.
- **Wände:** Anforderungen an den Brandschutz, Putz und Farbe. Für diese Arbeit nicht relevant
- **Decke:** siehe *Wände*.
- **Boden:** In den Betriebsräumen wird stets ein Doppelboden verlegt. Die Anforderungen sind aufgrund ihrer vielen individuellen Besonderheiten je Raum nicht leicht erfassbar und werden daher wie die Decken und Wände nicht berücksichtigt.
- **Türen:** Neben Anforderungen an den Brandschutz und Mindestmaße, steht hier auch, wie viele Türen ein Raum haben muss und in welche Räume diese Türen führen müssen. Dabei handelt es sich um ein Kriterium, welches über die *RAO* dargestellt werden kann.

- **Lüftung:** Hier werden Angaben zur nötigen Heizungs-, Lüftungs-, und Klimatechnik gemacht. Diese stellen Raumbedürfnisse dar, welche in der Regel durch andere Räume erfüllt werden müssen. Allerdings handelt es sich dabei meist um optionale Anforderungen, welche nur nach ermitteltem Bedarf vorhanden sein müssen. Dies erschwert eine allgemeine Regelprüfung und setzt individuelle Prüfungssätze für jedes einzelne Projekt voraus.
- **Raumausstattung:** Hier wird aufgezählt, welche Infrastruktur im entsprechenden Raum vorhanden sein soll. Dabei geht es zunächst um die Einbauten, also interne Anforderungen. Aber aus diesen lassen sich meist sehr direkt die entsprechend nötigen *Ressourcen* ableiten. Wenn z.B. geschrieben wird, dass eine normale sowie eine Not- und Sicherheitsbeleuchtung vorgeschrieben wird, dann muss der Raum mit allen drei Stromarten als einzelne *Ressourcen* versorgt werden. Allerdings ist die nötige Versorgung, vor allem bei den vielen Stromarten, nicht immer ersichtlich, oder die Anforderungen dürfen teilweise auch vor dem Zugang erfüllt werden, was eine Prüfung ohne genauere geometrische Kenntnisse erschwert.
- **Sonstiges:** Zuletzt werden Zusatzanmerkungen aufgeführt. Häufig wird hier festgehalten, dass durch den Raum keine betriebsfremden Installationen führen dürfen, oder ob eine Sprinklerung vorhanden sein darf oder nicht. Auch Angaben zur Lage im Gebäude, oder relativ zu anderen Räumen und wie oft der Raum ggf. vorhanden sein muss.

Wie in der Einleitung erwähnt, wurde der *RLK* als Ausgangsgrundlage für diese Arbeit verwendet, weshalb sich die Fragestellung gut auf die Anforderungen des *RLK* zurückführen lassen. Am Beispiel eines ausgewählten Plans des U-Bahnhofs Willibaldstraße wird im Folgenden gezeigt, dass die Ontologie und die weiteren entwickelten Abfragen und Umsetzungskonzepte dieser Arbeit erfolgreich anwendbar sind.

### 7.2.3 Bahnhof Willibaldstraße

Der Bahnhof *Willibaldstraße* liegt in der Gotthardstraße in Höhe der Kreuzung mit der Willibaldstraße und schließt mit einer weiteren Abstellanlage an die bestehende Anlage am Laimer Platz an. Er besteht aus drei Geschossen: dem Unterbahnsteig, der Bahnsteigebene und einem Sperrengeschoss. Von letzterem führen vier Aufgänge an die Oberfläche. Der Bahnhof besitzt einen Innenbahnsteig mit zwei Bahnsteigkanten und an beiden Bahnhofsköpfen Betriebsräume. Im Westkopf befindet sich ein Gleichrichterwerk für den folgenden Streckenabschnitt. Nach dem Bahnhof folgt eine Aufweitung für einen möglichen Abzweig einer neuen Linie nach Süden. Anschließend schwenkt die Trasse nach Norden und führt über den Bahnhof *Am Knie* nach *Pasing* zum Fernbahnhof, wo der Streckenabschnitt vorerst endet.

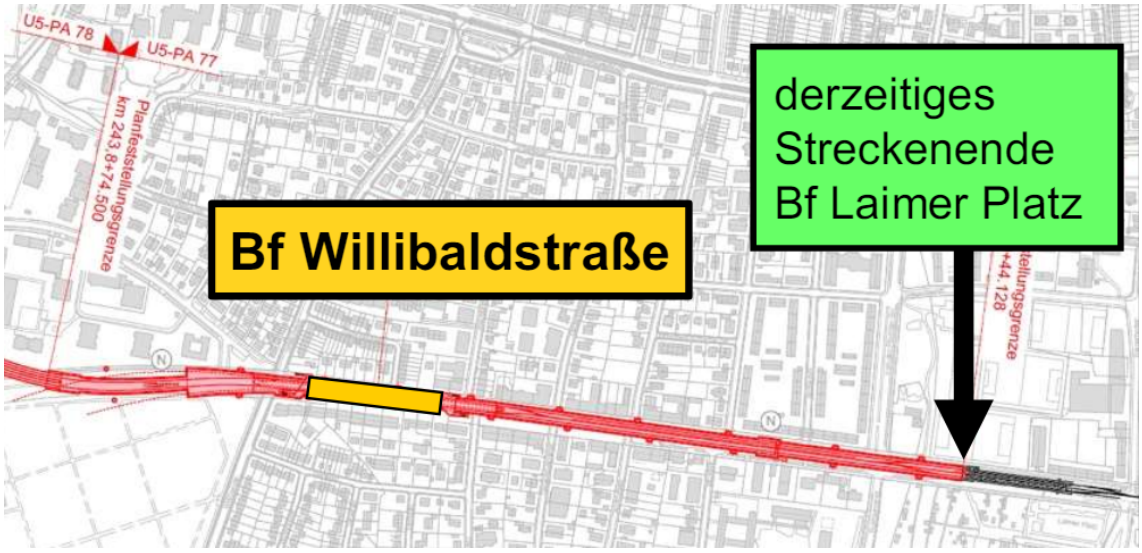



Abbildung 7.3: Lage des Bahnhofs *Willibaldstraße*. (Ausschnitt aus FRISCHEISEN (2023))

Die gewählte Plangrundlage für das im Folgenden verwendete Modell stammt vom Ende der Vorplanung und spiegelt damit einen Stand der Planung wieder, für den die Konzepte dieser Arbeit entwickelt wurden. Der Plan ist der Arbeit im Anhang B beigefügt und wird im Folgenden als *Plan WB* referenziert. Das *IFC*-Modell wurde in *Revit* modelliert und anschließend nach *IFC4 Reference View* exportiert. Fehler im Plan (z.B. fehlende Türen) wurden übernommen, um die Prüfung realitätsnaher zu gestalten.

### 7.2.4 Auswertung ausgewählter Regeln

Da sämtliche Konzepte dieser Arbeit bereits im ersten Beispiel erfolgreich demonstriert wurden, folgen nun nur ein paar ausgewählte Punkte und keine vollständige Prüfung.

#### Nachbarschaftsalgorithmus

Nach einer ersten Modellierung erkannte der Algorithmus zunächst nur 211 von 221 horizontalen und 136 von 136 vertikalen Nachbarschaften, sowie korrekterweise keine Überschneidungen und vollständigen Überlagerungen. Auch wenn dies bereits eine Erfolgsquote von 97 % darstellt, wurde das Ergebnis genauer ausgewertet und eine Wand identifiziert, welche für alle zehn Fehler verantwortlich war. Eine genaue Ursache konnte nicht festgestellt werden, außer, dass eine minimale Abweichung in der Lage bezüglich der Horizontalen bestand. Nachdem die Wand neu modelliert wurde, wurden keine Fehler mehr gefunden und der Test war somit zu 100 % erfolgreich.

#### Erreichbarkeit

Wenn man die Erreichbarkeit aller Räume mit der **Abfrage 2** prüft, so erhält man als Ergebnis, dass elf Räume nicht erreichbar sind. Bei genauerer Betrachtung lassen sich

zwei Fälle bei den elf Räumen unterscheiden. Der erste Fehler betrifft die drei Räume auf der Unterbahnsteig-Ebene, welche keinen ordentlich modellierten Zugang haben (Modellfehler). Die anderen Fehler kommen aus der Plangrundlage selbst. Dort sind nämlich insgesamt fünf Türen nicht eingezeichnet, was dazu führt, dass acht Räume nicht erreicht werden können. Fünf dieser Räume sind in der folgenden Abbildung zu sehen. Dort sieht man, dass der Löschwasserraum keinen Zugang besitzt und der Kiosk zwar mit seinem Lagerraum, seinem WC und dem Vorraum dazwischen verbunden ist, aber dieser Verbund wiederum keinen Zugang zu einem anderen Raum hat.

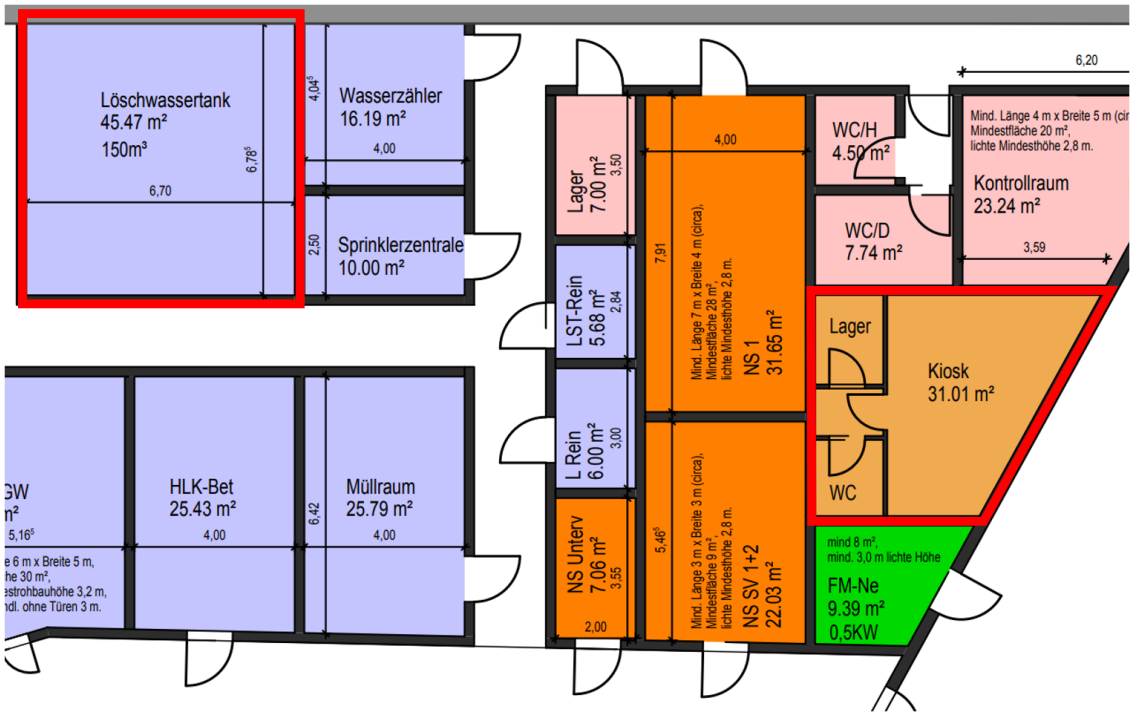



Abbildung 7.4: Ausschnitt aus dem *Plan WB*, auf dem zwei Räume bzw. Raumgruppen ohne Zugang rot markiert sind.

Prüft man explizit die Erreichbarkeit für *Fahrgäste* (modelliert als Instanz von *rao:Group*), dann ist die Prüfung erfolgreicher, denn alle Räume, die als öffentlicher Bereich deklariert wurden, sind vollständig zugänglich und verbunden.

### Gleichrichterwerk

Ein Gleichrichterwerk ist wie ein Stellwerk eine Raumgruppe innerhalb eines Bahnhofes. Ein Werk besteht aus mehreren Räumen, welche kompakt zusammenliegen sollen und eine getrennte Versorgung benötigen. In einem Gleichrichterwerk wird der, vom Umspannwerk kommenden, Wechselstrom zu 750 V Gleichstrom für den Fahrbetrieb der anliegenden Gleisabschnitte gewandelt. Dafür werden neben einem Raum für die Transformatoren (*GW-Trafo*) und einem für die Schalteinrichtungen (*GW-Schalt*), ein eigener Batterieraum für die Notstromversorgung der Raumgruppe, ein Kontrollraum (*GW-Meist*) und eine Werkstatt bzw. Ersatzteillager (*GW-Sp*) benötigt. Diese Räume sollten alle auf der Bahnsteigebene liegen und müssen alle über einen gemeinsamen Kabelkeller mit-

einander verbunden sein. In der folgenden Abbildung ist eine schematische Darstellung eines Gleichrichterwerks zu sehen. Dort sind auch die geforderten Zugangsbeziehungen zu sehen.

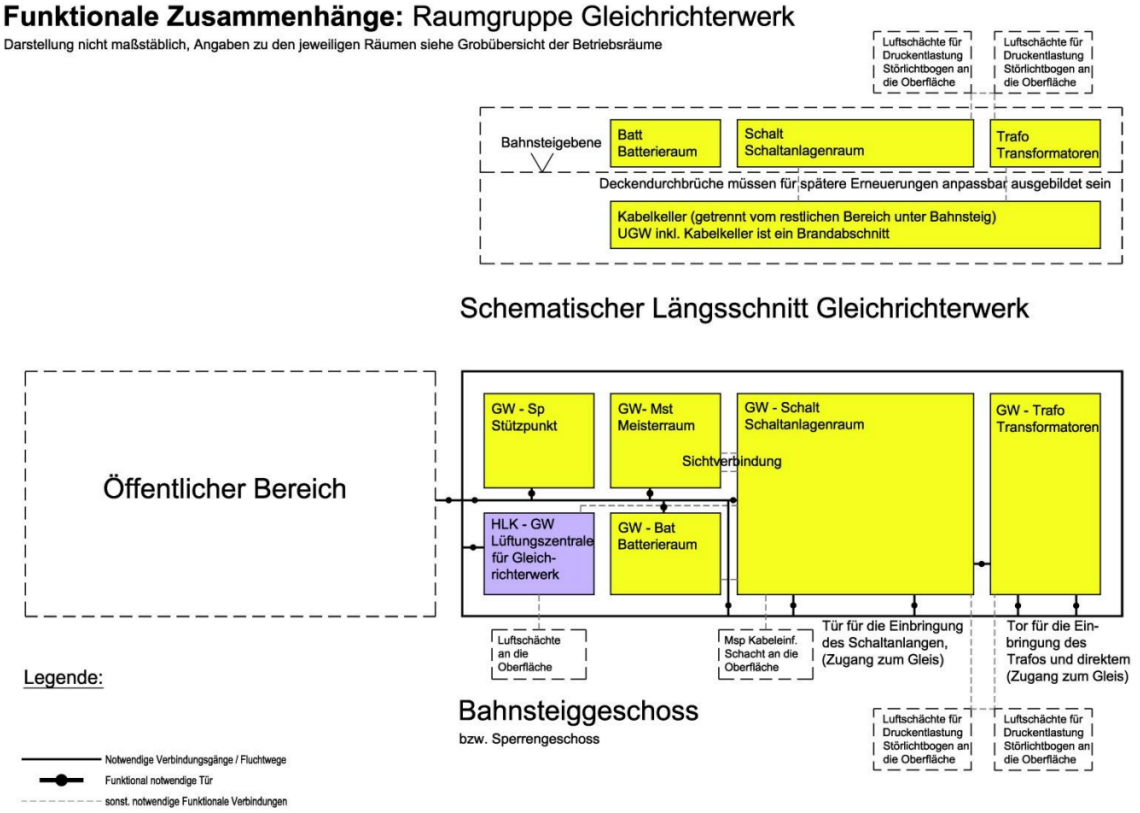



Abbildung 7.5: Schema eines Gleichrichterwerks mit allen Raumbeziehungen. (BAU-J & SWM, 2023)

Zusätzlich zu den abgebildeten Beziehungen gibt es im *RLK* weitere Anforderungen. Ausgewählte Teile davon wurden für diese Arbeit in einer *Turtle*-Datei formuliert (siehe Beispiel A.1 im Anhang).

Im Bahnhof *Willibaldstraße* befindet sich das Gleichrichterwerk im westlichen Bahnhofskopf, wie gefordert, auf Bahnsteigebene. Die Lüftungszentrale aus dem Schema befindet sich eine Etage darüber und wird in diesem Beispiel nicht weiter berücksichtigt.

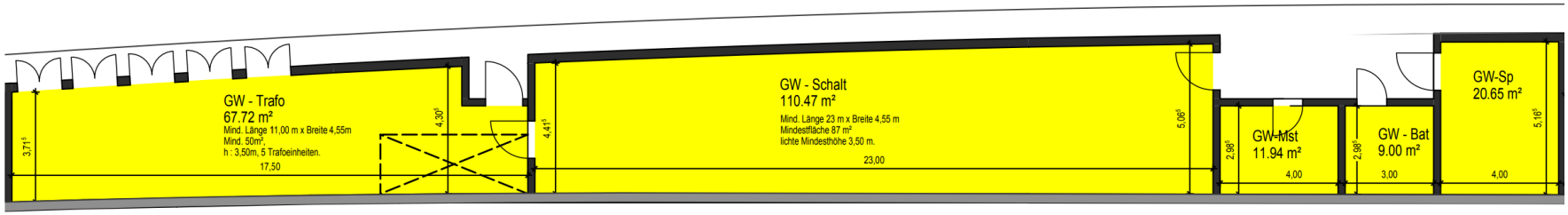



Abbildung 7.6: Lage des Gleichrichterwerks im Bahnhof Willibaldstraße (Ausschnitt aus *Plan WB*). Alle Räume werden nach Norden in Richtung Gleis 1 entfluchtet.

Die wichtigste Anforderung im RLK ist, dass alle Räume des Gleichrichterwerks zusammenhängen und speziell Meisterraum, sowie die Transformatoren unbedingt an den Schaltanlagenraum angrenzen. Für die Prüfung der Nachbarschaften kann **Abfrage 5** modifiziert werden. Hierbei zeigt diese simple Abfrage allerdings zwei Schwächen. Es haben alle Räume des Werks die Funktion *inst:GW* zugeordnet bekommen und statt dass die Abfrage prüft, ob jeder Raum mit *mindestens* einem anderen Raum mit dieser Funktion verbunden ist, erwartet die Abfrage, dass alle Räume mit allen anderen Räumen der Funktion verbunden sein müssen, was sie nicht sind. Beispielsweise sind Gleis 1 & 2 jeweils als *Gleis* und als *Fluchtraum* deklariert. Alle Räume des Gleichrichterwerks können korrekterweise zum Gleis 1 hin entfluchtet werden. Die Abfrage wirft aber aus, dass Gleis 2 ebenfalls eine Fluchtmöglichkeit ist und dort hin keine direkten Zugänge bestehen würden. Prüft man lediglich die Beziehung zwischen *GW-Mst, GW-Trafo* und *GW-Schalt*, dann erhält man als Ergebnis, dass diese korrekt verbunden sind. Auch ein weiteres Problem wird in dieser Abfrage nicht richtig erkannt. Zieht man *Abfrage 1* hinzu, stellt man fest, dass der Kabelkeller (*GW-Kab*) gar nicht definiert wurde und im Plan fehlt. Dieser Fall, dass eine Funktion gar nicht zugeordnet wurde, war in der **Abfrage 5** ebenfalls nicht berücksichtigt. Somit müsste die Abfrage, welche für das erste Beispiel korrekt funktioniert hat, hierfür verbessert werden. Prüft man statt der Nachbarschaft die Zugänge (*rao:hasAccessTo*), so liefert die Abfrage ebenfalls wieder korrekte Ergebnisse.

Ein weiterer prüfbarer Punkt sind die Mindestmaße. Diese Anforderungen wurde ebenfalls in den Beispielanforderungen (siehe A.1) aus dem *RLK* übernommen. Um das Ergebnis übersichtlich zu halten, wurde sich jeweils auf die Mindestgrundfläche und die Mindesthöhe beschränkt. Die **Abfrage 3** wurde für das folgende Ergebnis so modifiziert, dass nicht nur die fehlerhaften, sondern alle geprüften Werte ausgegeben werden. Für die Übersichtlichkeit wurden zudem die Spalten *Funktion* und *Raumname* in der Tabelle zusammengefasst, da sie sich nur um das Präfix unterscheiden und sonst von der Bezeichnung her identisch sind. Ebenso wurden alle leeren Spalten (für Werte mit exakter Übereinstimmung und für Maximalwerte) entfernt. Es ist in der Tabelle ersichtlich, dass alle Mindestmaße bis auf die Grundfläche des Batterieraums eingehalten werden. Der Batterieraum ist aber lediglich 0,3 m² zu klein, was in einer späteren Planungsphase leicht ausgebessert, oder auch mit einer Begründung als Ausnahme akzeptiert werden kann. Der fehlende Kabelkeller taucht auch in dieser Tabelle nicht auf.

Tabelle 7.6: Prüfung der Mindestanforderungen aller Räume des Gleichrichterwerks mit **Abfrage 3**.

| Raum | Raumname & Funktion | Eigenschaft | value | reqMin | Violated |
|---|---|---|---|---|---|
| inst:IfcSpace_3251 | GW-Schalt | inst:Area | 110,8 | 87 | false |
| inst:IfcSpace_3251 | GW-Schalt | inst:Height | 4,0 | 3,5 | false |
| inst:IfcSpace_3171 | GW-Bat | inst:Area | 8,7 | 9 | true |
| inst:IfcSpace_3171 | GW-Bat | inst:Height | 4,0 | 2,8 | false |
| inst:IfcSpace_3371 | GW-Trafo | inst:Area | 67,7 | 50 | false |
| inst:IfcSpace_3371 | GW-Trafo | inst:Height | 4,0 | 3,5 | false |
| inst:IfcSpace_3536 | GW-Mst | inst:Area | 11,8 | 10 | false |
| inst:IfcSpace_3536 | GW-Mst | inst:Height | 4,0 | 2,8 | false |
| inst:IfcSpace_3076 | GW-Sp | inst:Area | 20,6 | 12 | false |
| inst:IfcSpace_3076 | GW-Sp | inst:Height | 4,0 | 2,8 | false |

**Raumbedürfnisse**

Als letztes Beispiel wird die Erfüllung von Raumbeziehungen geprüft. Es wird untersucht, ob ausgewählte Ressourcen von den Verteilern zu jedem nachfragendem Raum möglich sind. Vereinfacht wurde definiert, dass alle Räume eine Stromversorgung brauchen (*inst:Strom*). Zudem benötigen alle Betriebsräume einen Netzwerkanschluss (*inst:Netzwerk*). Von der Mittelspannungsverteilung soll eine Mittelspannungsleitung (*inst:MS*) zu allen Niederspannungsräumen führen. Diese drei *Ressourcen* stehen exemplarisch für die vielen und komplexen Ressourcen, die in einem U-Bahnhof vorkommen. Mit diesen drei *Ressourcen* lassen sich aber die Fälle *großflächige Verteilung (Strom, Netzwerk)* und *punktuelle Verteilung (MS)* gut abdecken, sodass die wesentlichen Fälle gezeigt werden. Ressourcen dürfen nicht durch andere Betriebsräume geleitet werden (pauschal ohne Ausnahmen) und als Annahme dürfen ebenfalls keine Ressourcen durch den Bahnsteigraum und die Gleisräume gelegt werden. Dabei handelt es sich um den Raum über dem Bahnsteig, welcher für Fahrgäste benutzbar und einsehbar ist und die Raumvolumen entlang der Gleise inklusive der Betriebswege.

Führt man nun **Abfrage 4.1** aus, so erhält man als Ergebnis, dass je drei Räume keine Anbindung an *Strom* und *Netzwerk* haben und dass zwischen fünf Niederspannungsverteilungsräumen und den Mittelspannungstransformatoren keine Verbindung möglich ist. Bei den ersten zwei *Ressourcen* handelt es sich um die selben drei Räume. Davon liegen zwei an den Bahnhofsenden auf der Bahnsteigebene, wo sie nur noch an die Gleise und den vorletzten Raum angrenzen. Dies sind alles Räume die keine fremden Ressourcen durchlassen. Der dritte Raum ist ein Aufzugstechnikraum unter dem Bahnsteig, der laut Graph nur an den darüberliegenden Bahnsteigsraum angrenzt, welcher keine Ressourcen zu diesem durchlässt. Liest man mit **Abfrage 4.2** auch noch die Namen der fünf Niederspannungsräume aus, dann stellt man fest, dass alle Räume im Westkopf betroffen sind und keiner der Räume im Ostkopf, wo auch die Mittelspannungsräume liegen. Es sind also beide Bahnhofsköpfe getrennt. Bei der Niederspannung und beim Netzwerk ist das nicht aufgefallen, da es auf jeder Etage in jedem Kopf einen Verteiler gibt. Prüft man allerdings

ob die Räume für die Fernmeldetechnik, welche das Netzwerk an die umliegenden Räume verteilen, auch untereinander verbunden sind, also von Verteiler zu Verteiler statt zu den Empfängern, dann erkennt man das gleiche Problem wie bei der Mittelspannung.

Die Ursache dieser Probleme ist, dass der Kabelkeller unter der Bahnsteigebene nicht als Raum ausgewiesen ist und daher dem Graph nicht bekannt ist (vgl. Gleichrichterwerk). Definiert man nun den Unterbahnsteig im Modell und damit auch im Graph als expliziten Raum, der alle *Ressourcen* durchleiten darf, dann sind alle Fehler behoben. Der Kabelkeller reicht von einem Bahnhofsende zum anderen und liegt somit unter allen Räumen der Bahnsteigebene. Vereinfachend wird der Kabelkeller des Gleichrichterwerks nicht abgetrennt. Dadurch werden auch die äußersten Räume und der Aufzugsraum mit allen *Ressourcen* verbunden. Ebenso werden durch den Keller beide Bahnhofsköpfe bzgl. der *Ressourcen* miteinander verknüpft. Dadurch sind auch alle Netzwerkräume miteinander verbunden.

### 7.2.5 Zusammenfassung

Diese beispielhaften Prüfungen zeigen, dass auch reale Regelwerke wie der *RLK* durch die *Room Analysis Ontology* abgebildet und geprüft werden können. Die offensichtlichen Fehler im Plan (fehlende Türen und Kabelkeller) konnten identifiziert werden.

Es wurde aber auch festgestellt, dass in der Realität komplexere Abfragen nötig sind. Obwohl die Abfragen im ersten Beispiel funktioniert haben und das zweite Beispiel auch korrekt als Graph abgebildet wurde, wurden jetzt Fehler übersehen bzw. falsch erkannt. Da es sich bei den Abfragen aber lediglich um Beispiele handelt und nicht um die einzige Lösung, ist nicht davon auszugehen, dass die Ontologie verbesserungswürdig ist, sondern nur die Abfragen.

# Kapitel 8

# Fazit

In dieser Arbeit wurde eine Ontologie entwickelt, welche in der Lage ist Raumbeziehungen und -bedürfnisse darzustellen. Dabei können sowohl die vorhandenen Raumeigenschaften als auch die Anforderungen an die Räume beschrieben werden. Der Aufbau der Ontologie wurde anhand von *Competency Questions* hergeleitet. Anschließend wurde für jede dieser Fragen eine geeignete *SPARQL*-Abfrage entwickelt, welche beispielhaft die Erfüllung der Frage prüfen sollte. Dieser Satz an Abfragen wurde dann auf zwei Beispiel-Bauwerke angewendet. Es konnte dadurch gezeigt werden, dass die gestellten *Competency Questions* erfüllt werden.

Es hat sich herausgestellt, dass alle, in dieser Arbeit aufgestellten Fragen, rein mittels einem Graphen aus BOT und RAO, ohne weitere bautechnische Ontologien beantwortet werden können. Das heißt, es werden sonst nur grundlegende Ontologien wie *Schema, RDFS* oder *OWL* verwendet. Für die meisten Abfragen war der Graph aus dem eigens entwickelten Converter ausreichend und nur selten war eine Verknüpfung mit dem LBD-Graph nötig.

Mit dem Nachbarschaftsalgorithmus und den weiteren Algorithmen zur Extrahierung der Daten aus IFC konnte demonstriert werden, wie eine Verknüpfung zwischen *IFC*-Dateien und der *RAO* aussehen kann. Es wurde gezeigt, wie Raumnachbarschaften auf Basis bekannter Algorithmen berechnet werden können, getrennt für vertikale und horizontale Nachbarschaften. Dieser Algorithmus hat für die Beispiele dieser Arbeit fehlerfrei funktioniert. Allerdings gilt das nicht für beliebige Gebäude. Denn es wurden Vereinfachungen getroffen und zudem sind auch bereits einige Fehlinterpretationen bekannt. Zukünftig könnte man an dieser Stelle ansetzen und den Nachbarschaftsalgorithmus verbessern, sodass die bereits bekannten Probleme, welche in dieser Arbeit nicht mehr gelöst werden konnten (z.B. Eckbeziehungen und Toleranz), beseitigt werden und bestenfalls auch manche Anwendungsgrenzen wegfallen. Es ist allerdings hervorzuheben, dass der Algorithmus bereits für einen Großteil der vorkommenden Raumkonstellationen funktioniert und die Probleme lediglich Spezialfälle betreffen. Im Sinne einer automatisierten Anwendung, sollte aber eine möglicht geringe Fehlerquote angestrebt werden. Die aktuelle Version genügt aber, für gewöhnliche Raumstrukturen, wie in den Beispielen.

Ein weiterer optimierbarer Prozess sind die Abfragen. Die Qualität der Abfragen ist entscheidend für die Nutzbarkeit der Ergebnisse. Es wurde mit den Abfragen gezeigt, wie die Fragestellungen auf verschiedene Weisen ausgewertet werden können. Für eine praktische Umsetzung sind die Abfragen aber nur bedingt geeignet, da gezielt möglichst viele Ergebnis-Arten gezeigt wurden. Bei einer vollständigen Prüfung eines Regelwerks sollten alle Abfrage-Ergebnisse einen konstanten Stil haben (z.B. stets nur *Regeln einge-*

*halten* oder *nicht eingehalten* als Ergebnis). Allerdings hängen die Abfragen auch sehr von den gewählten Fragen, bzw. den zugrundeliegenden Daten und Anforderungen ab, weshalb eine allgemeingültige Optimierung nicht unbedingt anzustreben ist. Viel mehr sollte als nächstes ein ausführlicher Praxistest erfolgen, um sicherstellen zu können, dass die Ontologie auch für komplexere Szenarien geeignet ist.

Aktuell ist bei den Beispielen noch immer eine menschliche Interpretation der Ergebnisse nötig, um eventuell übersehene Fehler zu erkennen. Auch wurden die fiktiven bzw. vereinfachten Regelwerke bis jetzt händisch in einen *RDF*-Graph übersetzt. Diese Methode ist nicht nur langwierig, sondern birgt auch ein hohes Fehlerrisiko (z.B. durch Tippfehler oder vergessene Statements). Das nächste Ziel ist deshalb, ein reales Regelwerk vollständig zu übersetzen und dafür anschließend geeignete Abfragen zu entwickeln. In diesem Zuge könnte man untersuchen, wie weit sich diese Übersetzung automatisieren bzw. zumindest vereinfachen lässt.

Abschließend lässt sich sagen, dass in dieser Arbeit eine stabile Grundlage geschaffen wurde, um Raumbeziehungen zu erkennen und zu prüfen. Auf den Ergebnissen dieser Arbeit kann aufgebaut werden, um die gewonnenen Erkenntnisse zu festigen, die entwickelten Prozesse und Konzepte zu optimieren und vielleicht auch neue Anwendungsfälle für die *Room Analysis Ontology* zu finden.

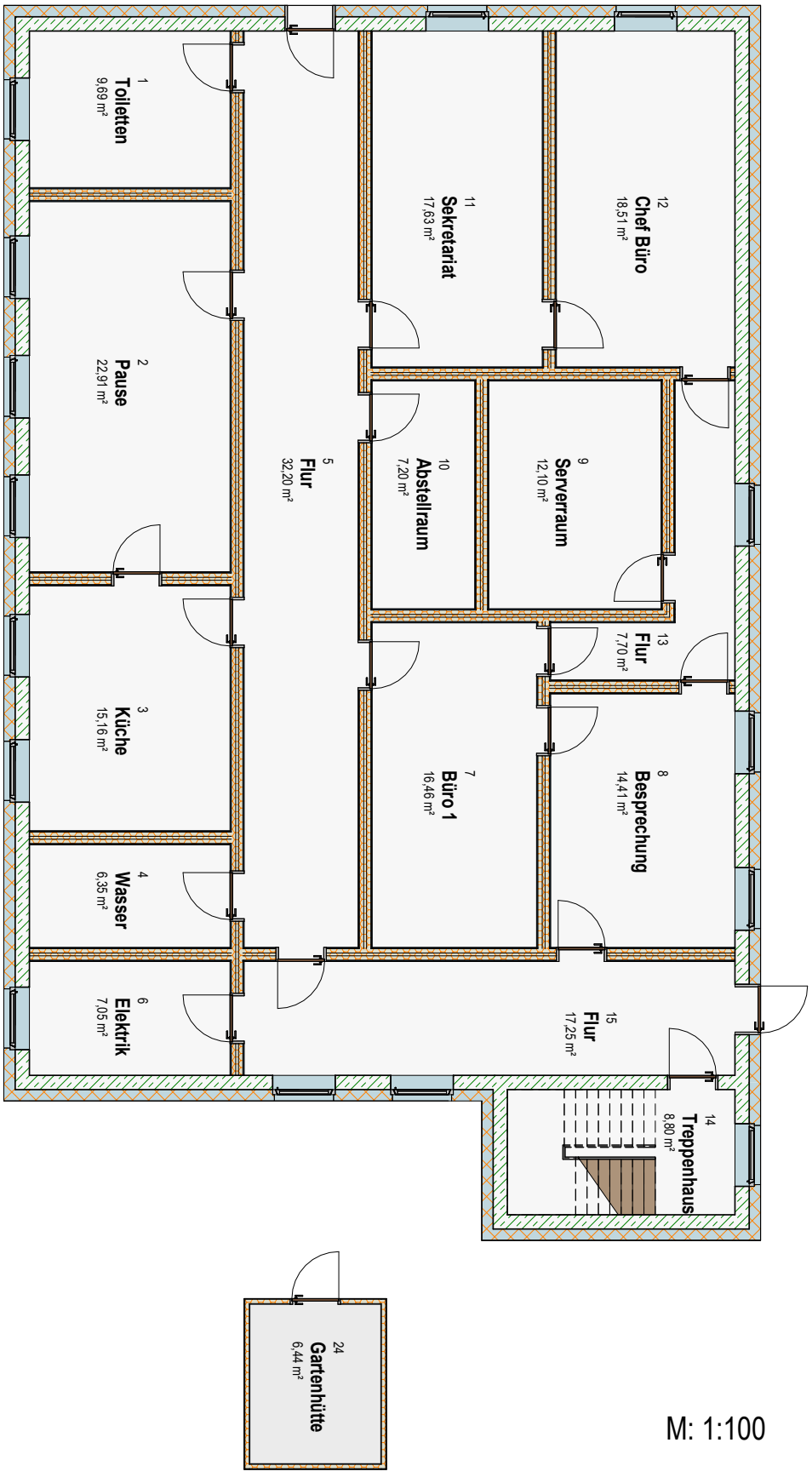
# Anhang A

# Pläne und Code-Beispiele

## A.1 Grundrisse Bürogebäude

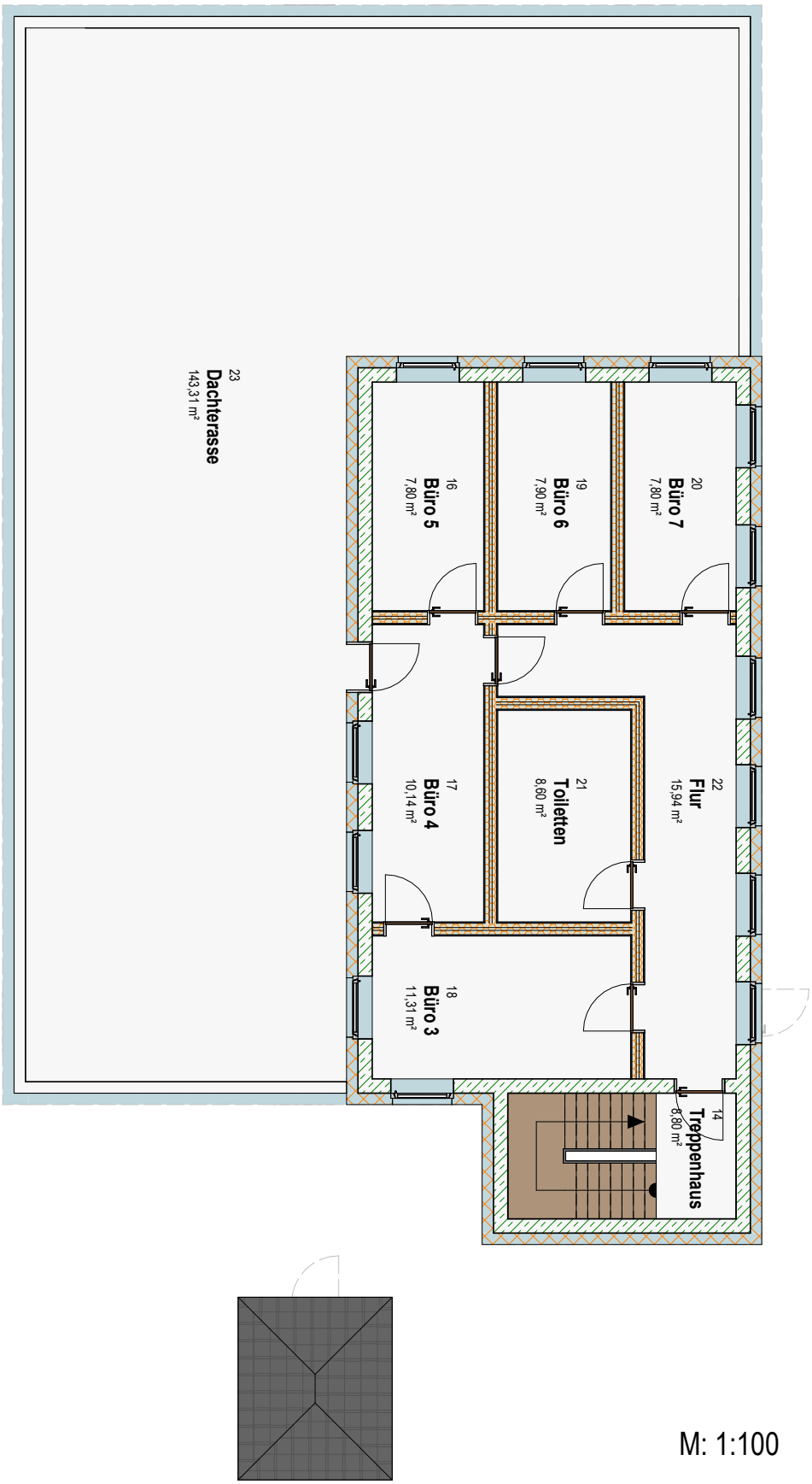
Es folgen die Grundrisse des im ersten Beispiel verwendeten Gebäudes.

# Beispielhaus, EG




M: 1:100

# Beispielhaus, 1. OG




M: 1:100

## A.2 Anforderungen Gleichrichterwerk

Das folgende Code-Beispiel sind ausgewählte Regeln aus dem *RLK* an ein Gleichrichterwerk im *Turtle*-Format.

Algorithmus A.1: Anforderungen an ein Gleichrichterwerk.

```
@prefix inst: <www.test.org/> .
@prefix rao: <http://www.semanticweb.org/ontologies/RoomAnalysis/> .
@prefix schema: <https://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# properties
inst:Height rdfs:subClassOf rao:GeometricProperty.
inst:Area rdfs:subClassOf rao:GeometricProperty .
inst:Volume rdfs:subClassOf rao:GeometricProperty .

# functions
inst:Gleis a rao:Function.
inst:Fluchtraum a rao:Function.

inst:GW a rao:Function;
    rao:isNextTo inst:Fluchtraum,
                 inst:GW;
    rao:hasAccessTo inst:Fluchtraum.

inst:GW-Schalt a rao:Function;
    rao:hasProperty inst:SchaltFläche,
                    inst:SchaltHöhe;
    rao:isNextTo rao:GW;
    rao:isAbove inst:GW-Kab;
    rao:hasAccessTo inst:Gleis.

inst:GW-Trafo a rao:Function;
    rao:hasProperty inst:TrafoFläche,
                    inst:TrafoHöhe;
    rao:isNextTo rao:GW-Schalt;
    rao:isAbove inst:GW-Kab;
    rao:hasAccessTo inst:GW-Schalt,
                    inst:Gleis.

inst:GW-Mst a rao:Function;
    rao:hasProperty inst:MstFläche,
                    inst:MstHöhe;
    rao:isNextTo rao:GW-Schalt;
    rao:isAbove inst:GW-Kab.

```

```
41
42 inst:GW-Sp a rao:Function;
43     rao:hasProperty inst:SpFläche,
44                     inst:SpHöhe;
45     rao:isNextTo inst:GW;
46     rao:isAbove inst:GW-Kab.
47
48 inst:GW-Bat a rao:Function;
49     rao:hasProperty inst:BatFläche,
50                     inst:BatHöhe;
51     rao:isNextTo inst:GW;
52     rao:isAbove inst:GW-Kab.
53
54 inst:GW-Kab a rao:Function.
55
56 # values
57 inst:SchaltFläche a inst:Area;
58     schema:minValue "87"^^xsd:double.
59
60 inst:TrafoFläche a inst:Area;
61     schema:minValue "50"^^xsd:double.
62
63 inst:MstFläche a inst:Area;
64     schema:minValue "10"^^xsd:double.
65
66 inst:SpFläche a inst:Area;
67     schema:minValue "12"^^xsd:double.
68
69 inst:BatFläche a inst:Area;
70     schema:minValue "9"^^xsd:double.
71
72 inst:SchaltHöhe a inst:Height;
73     schema:minValue "3.5"^^xsd:double.
74
75 inst:TrafoHöhe a inst:Height;
76     schema:minValue "3.5"^^xsd:double.
77
78 inst:MstHöhe a inst:Height;
79     schema:minValue "2.8"^^xsd:double.
80
81 inst:SpHöhe a inst:Height;
82     schema:minValue "2.8"^^xsd:double.
83
84 inst:BatHöhe a inst:Height;
85     schema:minValue "2.8"^^xsd:double.
```

# Anhang B

# Beigefügte Elemente

Folgende Elemente werden zusammen mit dieser Arbeit als Anhang abgegeben:

## Room Analysis Ontology

- *Room Analysis Ontology* als *RAO.rdf*
- Dokumentation zur *RAO* (*.pdf*)

## Python-Scripte

Nachbarschaftsalgorithmus und Graph-Konvertierung als eine *Visual Studio*-Solution: *Read IfcSpace Polygons.zip*

## Beispielmodelle

### Bürogebäude

- Revit 2024-Modell (.rvt)
- IFC-Modell (.ifc)
- Graphen aus eigenem Converter, ifc2LBD und Anforderungen (.ttl)
- Parameter- und Exportkonfigurationen für Revit (.txt)
- Unterverzeichnis mit Listen von Ressourcen, Gruppen und Funktionen für Converter

### U-Bahnhof Willibaldstraße

Einzelne Datensätze für die Anwendungsfälle (AwF) *Gleichrichterwerk (GW)* und *Raumbedürfnisse (Mit Kabelkeller & ohne)*

- Revit 2024-Modell (.rvt)
- IFC-Modell (.ifc); je AwF
- Graphen aus eigenem Converter und Anforderungen (.ttl); je AwF

## Datengrundlagen LHM

Planungsgrundlagen aus dem Baureferat der Landeshauptstadt München (LHM):

- Richtlinienkatalog U-Bahn (.pdf)
- Grundrisse Willibaldstraße M 1:250, (2017); referenziert als *Plan WB* (.pdf)

# Literatur


ALLEMANG, D. (2011). *Semantic Web for the working ontologist: Effective modeling in RDFS and OWL* (2nd ed.). Morgan Kaufmann Publishers/Elsevier.

AUTODESK. (2024). Autodesk Revit | Preise ansehen & offizielle Revit-Software kaufen. Verfügbar 19. Juni 2024 unter https://www.autodesk.de/products/revit/

BARBAU, R., KRIMA, S., RACHURI, S., NARAYANAN, A., FIORENTINI, X., FOUFOU, S., & SRIRAM, R. D. (2012). OntoSTEP: Enriching product model data using ontologies. *Computer-Aided Design*, *44*(6), 575–590. https://doi.org/10.1016/j.cad.2012.01.008

BAU-J & SWM. (2023). Richtlinienkatalog U-Bahn: Für die Ausrüstung und die Gestaltung der U-Bahnhöfe, ihrer Betriebsräume und Einrichtungen.

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*(1), 89–101. https://doi.org/10.1017/S0890060409000122

BERG, M., CHEONG, O., VAN KREVELD, M., & OVERMARS, M. (2008). *Computational geometry: Algorithms and applications* (3. ed.). Springer.

BERNERS-LEE, T. (2006). Linked Data - Design Issues. Verfügbar 21. Mai 2024 unter https://www.w3.org/DesignIssues/LinkedData.html

BERNERS-LEE, T., HENDLER, J., & LASSILA, O. (2001). The Semantic Web. In *Scientific American Vol. 284* (S. 34–43).

BIZER, C., HEATH, T., & BERNERS-LEE, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22. https://doi.org/10.4018/jswis.2009081901

BONDUEL, M., ORASKARI, J., PAUWELS, P., VERGAUWEN, M., & KLEIN, R. (2018). The IFC to linked building data converter: current status. *6th International Workshop on Linked Data in Architecture and Construction*, 34–43.

BORRMANN, A., KÖNIG, M., KOCH, C., & BEETZ, J. (2021). *Building Information Modeling*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4

BRICK. (2024). Home - BrickSchema. Verfügbar 28. Mai 2024 unter https://brickschema.org/

BUILDINGSMART INTERNATIONAL. (2023). 5.4.3.59 IfcRelSpaceBoundary - IFC 4.3.2 Documentation. Verfügbar 4. Mai 2024 unter https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcRelSpaceBoundary.htm

BUILDINGSMART INTERNATIONAL. (2019). Industry Foundation Classes (IFC) - buildingSMART International. Verfügbar 7. Mai 2024 unter https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/

BUILDINGSMART INTERNATIONAL. (2024). IFC Schema Specifications - buildingSMART Technical. Verfügbar 7. Mai 2024 unter https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/

DATACOMP. (2024). BIMvision kostenloser IFC Model Viewer | BIMvision. Verfügbar 19. Juni 2024 unter https://bimvision.eu/de/

DAUM, S., & BORRMANN, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. *Advanced Engineering Informatics*, *28*(4), 272–286. https://doi.org/10.1016/j.aei.2014.06.001

DE GIACOMO, G., & LENZERINI, M. (1996). TBox and ABox Reasoning in Expressive Description Logics. *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, *1996*, 37–48.

de FARIAS, T. M., ROXIN, A., & NICOLLE, C. (2015). IfcWoD, Semantically Adapting IFC Model Relations into OWL Properties. http://arxiv.org/pdf/1511.03897

DIN 1356-1: 2024-04. *Bauzeichnung - Teil 1: Grundregeln der Darstellung*. 2024-04.

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

DUCHARME, B. (2013). *Learning SPARQL: Querying and updating with SPARQL 1.1* (Second edition). O'Reilly.

ERICSON, C. (2005). *Real-time collision detection*. Elsevier Morgan Kaufmann. http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10169791

FRISCHEISEN, F. (2023). Planung im Tunnelbau aus Sicht des Bauherren anhand von Beispielen zu Münchner Infrastrukturprojekten. In *Planung im Tunnelbau*. Technische Universität München.

HAINES, E. (1994). Point in polygon strategies. In *Graphics Gems IV* (S. 24–46). Academic Press Professional, Inc.

HESSE, W. (2005). Ontologie(n). Verfügbar 25. Mai 2024 unter https://gi.de/informatiklexikon/ontologien

HOWARD, R. (2017). RDF Triple Stores vs. Labeled Property Graphs: What's the Difference? Verfügbar 28. Mai 2024 unter https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/

IFCOPENSHELL. (2024). IfcOpenShell - The open source IFC toolkit and geometry engine. Verfügbar 3. Mai 2024 unter https://ifcopenshell.org/

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

ISO/IEC 9075-1: 2023-06. *Information technology — Database languages SQL Part 1: Framework (SQL/Framework)*. 2023-06.

KRÖTZSCH, M. (2012). OWL 2 Profiles: An Introduction to Lightweight Ontology Languages. In T. EITER (Hrsg.), *Reasoning web* (S. 112–183). Springer.

LANGENHAN, C., WEBER, M., LIWICKI, M., PETZOLD, F., & DENGEL, A. (2013). Graph-based retrieval of building information models for supporting the early design

stages. *Advanced Engineering Informatics*, *27*(4), 413–426. https://doi.org/10.1016/j.aei.2013.04.005

LBD-CG. (2022). GitHub - w3c-lbd-cg/ontologies: A list of ontologies related to Linked Building Data. Verfügbar 28. Mai 2024 unter https://github.com/w3c-lbd-cg/ontologies

LBD-CG. (2024). Linked Building Data Community Group. Verfügbar 19. Mai 2024 unter https://www.w3.org/community/lbd/

MAZAIRAC, W., & BEETZ, J. (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics*, *27*(4), 444–456. https://doi.org/10.1016/j.aei.2013.06.001

MÖLLER, T., & TRUMBORE, B. (1997). Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools*, *2*(1), 21–28. https://doi.org/10.1080/10867651.1997.10487468

MUNDANI, R.-P. (2019). Bau- und Umweltinformatik 2: Teil 3: Graphentheorie (1).

MUSEN, M. A. (2015). The Protégé Project: A Look Back and a Look Forward. *AI matters*, *1*(4), 4–12. https://doi.org/10.1145/2757001.2757003

NIKNAM, M., & KARSHENAS, S. (2017). A shared ontology approach to semantic representation of BIM data. *Automation in Construction*, *80*, 22–36. https://doi.org/10.1016/j.autcon.2017.03.013

OGC. (2024). OGC GeoSPARQL - A Geographic Query Language for RDF Data. Verfügbar 28. Mai 2024 unter https://docs.ogc.org/is/22-047r1/22-047r1.html#

OHORI, K. A. (2021). Boundary representation: Lesson 1.2. Verfügbar 4. Mai 2024 unter https://3d.bk.tudelft.nl/courses/backup/geo1004/2020/data/handout1.2.pdf

ONTOTEXT. (2024a). Graph Path Search — GraphDB 10.5 documentation. Verfügbar 13. Juni 2024 unter https://graphdb.ontotext.com/documentation/10.5/graph-path-search.html?

ONTOTEXT. (2024b). GraphDB Downloads and Resources. Verfügbar 28. Mai 2024 unter https://graphdb.ontotext.com/

ORASKARI, J., BONDUEL, M., MCGLINN, K., PAUWELS, P., PRIYATNA, F., WAGNER, A., KUKKONEN, V., STEYSKALAND, S., & LEHTONEN, J. (2024). IFCtoLBD. Verfügbar 29. April 2024 unter https://github.com/jyrkioraskari/IFCtoLBD/

PAUWELS, P., KRIJNEN, T., TERKAJ, W., & BEETZ, J. (2017). Enhancing the ifcOWL ontology with an alternative representation for geometric data. *Automation in Construction*, *80*, 77–94. https://doi.org/10.1016/j.autcon.2017.03.001

PAUWELS, P., & ROXIN, A. (2016). SimpleBIM: From full ifcOWL graphs to simplified building graphs. *Proceedings of the 11th European Conference on Product and Process Modelling*.

PAUWELS, P., & TERKAJ, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, *63*, 100–133. https://doi.org/10.1016/j.autcon.2015.12.003

PREIDEL, C., DAUM, S., & BORRMANN, A. (2017). Data retrieval from building information models based on visual programming. *Visualization in Engineering*, *5*(1). https://doi.org/10.1186/s40327-017-0055-0

PYTHON.ORG. (2024). Welcome to Python.org. Verfügbar 3. Mai 2024 unter https://www.python.org/

RASMUSSEN, M. H., HVIID, C. A., KARLSHØJ, J., & BONDUEL, M. (2018). Managing space requirements of new buildings using linked building data technologies. In J. KARLSHOJ & R. SCHERER (Hrsg.), *eWork and eBusiness in Architecture, Engineering and Construction* (S. 399–406). CRC Press. https://doi.org/10.1201/9780429506215-50

RASMUSSEN, M. H., & LEFRANÇOIS, M. (2018). Ontology for Property Management. Verfügbar 28. Mai 2024 unter https://w3c-lbd-cg.github.io/opm/

RASMUSSEN, M. H., LEFRANÇOIS, M., SCHNEIDER, G. F., & PAUWELS, P. (2020). BOT: The building topology ontology of the W3C linked building data group. *Semantic Web*, *12*(1), 143–161. https://doi.org/10.3233/SW-200385

RASMUSSEN, M. H., PAUWELS, P., HVIID, C. A., & KARLSHØJ, J. (2017). Proposing a Central AEC Ontology That Allows for Domain Specific Extensions. *Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction*, 237–244. https://doi.org/10.24928/JC3-2017/0153

RASMUSSEN, M. H., PAUWELS, P., LEFRANÇOIS, M., & SCHNEIDER, G. F. (2021). Building Topology Ontology (LINKED BUILDING DATA COMMUNITY, Hrsg.). Verfügbar 12. Februar 2024 unter https://w3c-lbd-cg.github.io/bot/

RDFLIB TEAM. (2023). rdflib 7.0.0 — rdflib 7.0.0 documentation. Verfügbar 3. Mai 2024 unter https://rdflib.readthedocs.io/en/stable/index.html

SOLIBRI. (2024). Solibri | BIM software for architects, engineers and construction. Verfügbar 19. Mai 2024 unter https://www.solibri.com/

TAUSCHER, E., BARGSTÄDT, H.-J., & SMARSLY, K. (2016). Generic BIM queries based on the IFC object model using graph theory. *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*.

TECLAW, W., RASMUSSEN, M. H., LABONNOT, N., ORASKARI, J. T., & HJELSETH, E. (2023). The semantic link between domain-based BIM models. https://doi.org/10.18154/RWTH-2023-10332

TERKAJ, W., & ŠOJIĆ, A. (2015). Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology. *Automation in Construction*, *57*, 188–201. https://doi.org/10.1016/j.autcon.2015.04.010

VDI 2552 BLATT 1: 2020-07. *Building Information Modeling: Grundlagen*. 2020-07, Juli.

W3C. (2012). OWL 2 Web Ontology Language Primer (Second Edition). Verfügbar 25. Mai 2024 unter https://www.w3.org/TR/owl2-primer/

W3C. (2013). SPARQL 1.1 Query Language. Verfügbar 29. Mai 2024 unter https://www.w3.org/TR/sparql11-query/

W3C. (2014a). RDF 1.1 Primer. Verfügbar 21. Mai 2024 unter https://www.w3.org/TR/rdf11-primer/

W3C. (2014b). RDF Schema 1.1. Verfügbar 25. Mai 2024 unter https://www.w3.org/TR/rdf-schema/

W3C. (2024). SPARQL 1.2 Query Language. Verfügbar 29. Mai 2024 unter https://www.w3.org/TR/sparql12-query/

WAGNER, A., BONDUEL, M., PAUWELS, P., & RÜPPEL, U. (2020). Representing construction-related geometry in a semantic web context: A review of approaches. *Automation in Construction*, *115*, 103–130. https://doi.org/10.1016/j.autcon.2020.103130

ZHU, J., WU, P., & LEI, X. (2023). IFC-graph for facilitating building information access and query. In *Automation in Construction*. https://doi.org/10.1016/j.autcon.2023.104778

# Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbst ändig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

---

Ort, Datum, Unterschrift