

Simulation von Gebäude-Anlagentechnik in der BIM-basierten Planung energieeffizienter Gebäude

Anbindung von Modelica-Modellen an OpenBIM-IFC-Modelle mittels semantischer Technologien

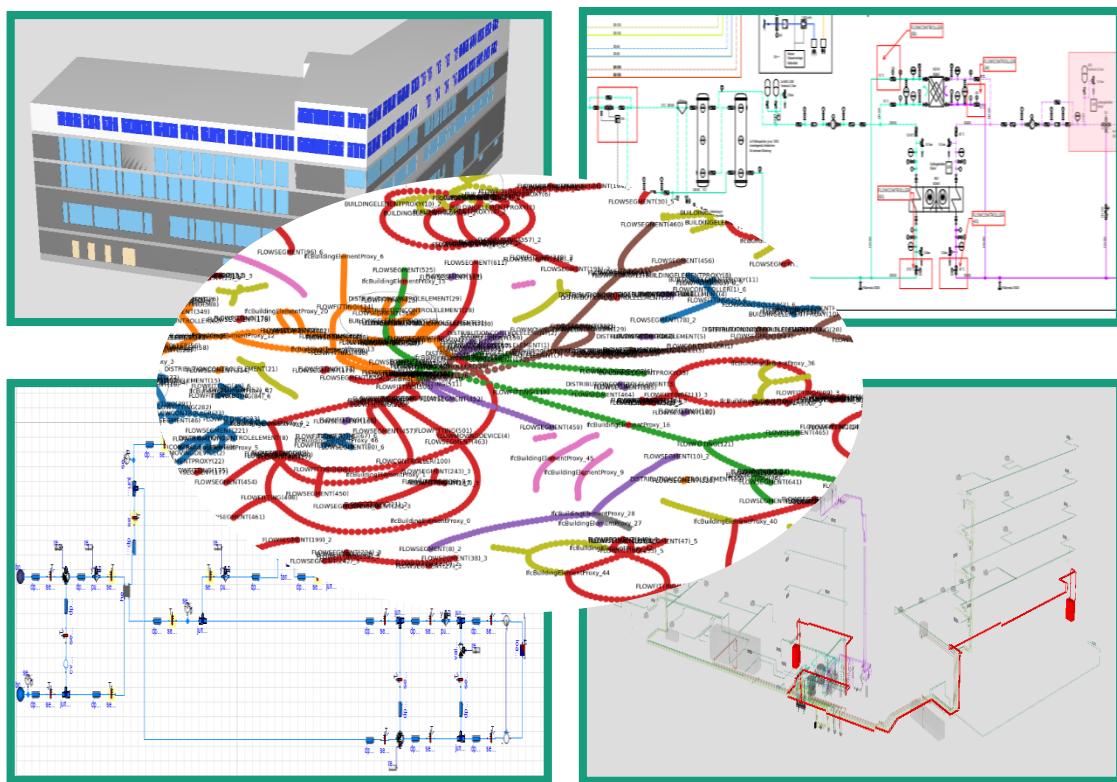
Dissertation

vorgelegt

an der Fakultät Bauingenieurwesen der TU Dresden

von Dipl.-Ing. Elisabeth Eckstädt

zur Erlangung des akademischen Grads Doktoringenieur



August 2024

Kurzinhaltsverzeichnis

1	Motivation, Stand von Wissenschaft und Technik	23
1.1	Motivation	24
1.2	Status Quo BIM.....	35
1.3	Status Quo der Simulation in der Gebäudeenergiedomäne	45
1.4	Status Quo des Zusammenwirkens von BIM und Simulation	48
1.5	Forschungsbedarf, Ableitung der Aufgabenstellung	52
1.6	Gliederung dieser Arbeit	54
2	Methodische und Technische Grundlagen.....	55
2.1	IFC.....	56
2.2	Modelica.....	75
2.3	Semantische Technologien.....	84
3	Methodischer Ansatz	93
3.1	Lösungsarchitektur des MO-x-IFC-Toolset	94
3.2	Vorteile durch die Verwendung von Wissensgraphen	95
3.3	Anforderungen an das zu entwickelnde Toolset	97
3.4	Überblick über die MO-x-IFC-Konverter	97
3.5	Schematizer (Konverter 2).....	100
3.6	MoOnt Modelica-Ontologie	107
3.7	Library-Ontologien	110
3.8	SemTran Semantic Translator (Konverter 4)	112
3.9	TTL2MO (Konverter 5)	119
3.10	MoTTL-Transcriptor (Konverter 7).....	120
3.11	ProTran Procedural Translator (Konverter 9).....	123
3.12	Voluminizer (Konverter 10)	124
4	Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web	125
4.1	Anforderungen an die Implementierung.....	126
4.2	Übersicht über die Konverter des MO-x-IFC-Toolsets.....	126
4.3	Details zu den einzelnen Konvertern MOXIK.....	127
5	Experimente	143
5.1	UseCase 1: BIM2SIM.....	144
5.2	UseCase 2: SIM2BIM.....	164
5.3	Vergleichende Auswertung zwischen Semantic und Procedural Translation	170
6	Schlussfolgerungen und kritische Bewertung	173
6.1	Prüfung und Diskussion der Thesen	174
6.2	Limitationen	175
6.3	Mehrwert gegenüber bestehenden Lösungen.....	175
6.4	Ausblick	176

Kurzzusammenfassung¹

Die Einbindung von Anlagensimulation in den Planungsprozess von technischer Gebäudeausrüstung trägt zu einem energieeffizienteren Betrieb von Gebäuden bei. Eine wesentliche Hürde bei der Anwendung der Simulation als Planungswerkzeug ist der Aufwand für die Modellerstellung. Diese Arbeit beschäftigt sich mit der automatisierten Modellerstellung basierend auf BIM (Building Information Management)-Modellen, um diesen Aufwand zu verringern. Weiterhin wird die Nachnutzung erstellter Simulationsmodelle durch die Erstellung von BIM-Modellen betrachtet.

Im Rahmen der Arbeit wurden Werkzeuge für die bidirektionale Verbindung zwischen BIM- und Simulationsmodell erstellt. Die Betrachtungen fokussieren sich dabei auf das Simulationsformat Modelica und das open-BIM-Format IFC. Die implementierten Werkzeuge basieren auf einer Analyse der Ausdrucksmächtigkeit der gewählten Formate (IFC und Modelica) und den Informationsanforderungen des ausgewählten Simulationsszenarios in den beiden UseCases BIM2SIM und SIM2BIM.

Für die Modellübertragung wurden Wissensgraphen für die intermediäre Wissensrepräsentation verwendet. Dies ermöglicht die Anwendung von in der IT bereits entwickelten semantischen Technologien für die Übersetzung zwischen BIM und Simulation. Weiterhin bieten die Wissensgraphen Anknüpfungspunkte für Anwendungsfälle die weit über die in dieser Arbeit untersuchten UseCases hinaus gehen.

Für beide betrachteten Anwendungsfälle wurde der experimentelle Nachweis erbracht, dass die automatisierte Modellerstellung soweit funktioniert, dass sie eine deutliche Zeiter spart in der praktischen Anwendung bringt. Für die experimentelle Untersuchung wurden dabei bewusst keine artifiziellen Minimalbeispiele verwendet, sondern im Sinne eines „Feldversuchs“ Modelle aus realen Projekten verwendet.

Im Ergebnis dieser Arbeit stehen Werkzeuge zur Verfügung, die eine Integration der Simulation in den Planungsprozess deutlich erleichtern:

- Sie ermöglichen die Bearbeitung einzelner Planungsschritte im jeweils geeigneteren Werkzeug (CAD-Autorensoftware bzw. BIM-Viewer oder Simulationsumgebung) bei gleichzeitiger Gewährleistung der inhaltlichen Integrität beider beteiligten Modelle.
- Es besteht erstmals die Möglichkeit für die Anlagentechnik in nennenswertem Umfang Modelica-Modelle aus BIM-Modellen zu erstellen.
- Die Erzeugung von weiterbearbeitbaren BIM-Modellen aus Modelica-Modellen wird in dieser Arbeit erstmals gezeigt.
- Die in der Arbeit durchgeführten Untersuchungen zur Nutzung des IFC-Formats zum Abspeichern semantisch reicher Schemata können auch in anderen Kontexten – unabhängig von der Simulation - Mehrwerte generieren.

¹ Eine Zusammenfassung der einzelnen Kapitel findet sich jeweils auf den Seiten 23, 55, 92, 125, 143 und 172.

English Abstract

The integration of plant simulation into the design process for technical building equipment contributes to more energy-efficient operation of buildings. A major hurdle in the utilisation of simulation as a design tool is the expense involved in model creation. This thesis deals with automated creation of models based on BIM (Building Information Management) models in order to reduce this effort. Furthermore, the utilisation of created simulation models through the creation of BIM models is considered.

Tools for the bidirectional link between BIM and simulation models were created as part of the work. The considerations focus on the simulation format Modelica and the open BIM format IFC. The implemented tools are based on an analysis of the expressive capabilities of the selected formats (IFC and Modelica) and the information requirements of the selected simulation scenario in the two use cases BIM2SIM and SIM2BIM.

Knowledge graphs for the intermediate knowledge representation were used for the model transfer. This enables the application of semantic technologies already developed in IT for the translation between BIM and simulation. Furthermore, the knowledge graphs provide starting points for use cases that go far beyond the use cases analysed in this thesis.

Experimental proof was provided for both use cases considered that automated model creation works to an extent that it significantly saves time in practical application. Deliberately no artificial minimal examples were used for the experimental investigation, but models from real projects were used in the spirit of a 'field test'.

As a result of this thesis, tools are available that significantly simplify the integration of simulation into the design process:

- They enable the processing of individual design steps in the more suitable tool (CAD authoring software or BIM viewer or simulation environment) while at the same time ensuring the integrity of the content of both models involved.
- For the first time, it is possible to create a significant amount of Modelica models from BIM models for plant engineering.
- The creation of processable BIM models from Modelica models is demonstrated for the first time in this thesis.
- The investigations carried out in the thesis on the use of the IFC format for storing semantically rich diagrams can also generate added value in other contexts - independent of simulation.

Danksagung

Diese Arbeit entstand 2019 bis 2024 in meiner Zeit als wissenschaftliche Mitarbeiterin am Fraunhofer IIS/EAS in Dresden. Sie wurde innerhalb der Projekte FMI4BIM (gefördert vom BMWI unter Kennzeichen: 03ET1603A) und iECO (gefördert vom BMWK 68GX21011D) erarbeitet. Weiterhin erhielt ich eine Förderung der Fraunhofer Gesellschaft im sog. Talenta-Programm. Die ersten Ansätze meiner Dissertation entstanden in den Jahren 2009 bis 2014 in meiner Zeit als wissenschaftliche Mitarbeiterin am Lehrstuhl für Gebäudeenergetik und Wärmeversorgung der TU Dresden. In dieser Zeit erhielt ich zusätzlich ein Promotionsstipendium der Stiftung der deutschen Wirtschaft. Ich bedanke mich bei den Fördermittelgebern für die Möglichkeit diese Arbeit zu erstellen.

Eine wesentliche Voraussetzung für die erzielten Ergebnisse waren die zahlreichen open source Software-Tools, die ich verwenden konnte. Ich bedanke mich bei deren Autoren und Institutionen für die Bereitstellung. Eine Auflistung findet sich in Tab. 54 auf S. 183.

Ausführliches Inhaltsverzeichnis

Kurzinhalsverzeichnis	3
Kurzzusammenfassung	4
English Abstract	5
Danksagung.....	6
Ausführliches Inhaltsverzeichnis.....	7
Abbildungsverzeichnis	12
Tabellenverzeichnis.....	14
Verzeichnis von Quelltext- und sonstigen Dateiausschnitten	16
Abkürzungsverzeichnis.....	17
Glossar	18
1 Motivation, Stand von Wissenschaft und Technik	23
1.1 Motivation	24
1.1.1 Gebäude im Kontext des Klimawandels	24
1.1.2 Planungsprozess der technischen Gebäudeausrüstung	25
1.1.3 Rolle von BIM.....	31
1.1.4 Simulieren im Planungsprozess	32
1.1.1.1 Anwendungsfälle.....	32
1.1.4.1 Mehrwerte.....	33
1.1.4.2 Hemmnisse	34
1.2 Status Quo BIM.....	35
1.2.1 Normung	35
1.2.1.1 Datenstandards	35
1.2.1.2 Modellstandards.....	36
1.2.1.3 Methoden- und Managementstandards	38
1.2.2 Softwarewerkzeuge für die Bearbeitung von IFC-Modellen.....	39
1.2.2.1 Systematisierung der Werkzeuge.....	39
1.2.2.2 IFC-Bibliotheken	40
1.2.2.3 IFC-Viewer	41
1.2.3 Forschung	42
1.2.3.1 Multimodelle, (distributed) Digital Twin	42
1.2.3.2 BIM und KI	43
1.2.3.3 Ontologien im Bereich Anlagentechnik, Regelungstechnik, Messdaten	43
1.3 Status Quo der Simulation in der Gebäudeenergiedomäne	45
1.3.1 Arten von Simulationsverfahren.....	45
1.3.2 Simulationswerkzeuge	46
1.3.3 Forschung	48
1.4 Status Quo des Zusammenwirkens von BIM und Simulation.....	48
1.4.1 BIM2SIM	48
1.4.2 SIM2BIM	50
1.4.3 Ontologien für Modelica-Modelle	50
1.5 Forschungsbedarf, Ableitung der Aufgabenstellung	52
1.5.1 UseCases	53
1.5.2 Thesen	53
1.6 Gliederung dieser Arbeit	54

2	Methodische und Technische Grundlagen	55
2.1	IFC.....	56
2.1.1	Allgemeines	56
2.1.2	IFC-Datenschemata	56
2.1.3	IFC-STEP-Dateien	58
2.1.4	Wesentliche abstrakte Entitäten zur Datenverwaltung	59
2.1.4.1	IfcRoot	59
2.1.4.2	IfcObject	59
2.1.4.3	IfcProduct	60
2.1.4.4	IfcElement	61
2.1.5	Entitäten mit physikalischer Entsprechung	61
2.1.5.1	Bauteile	61
2.1.5.2	Anlagentechnik.....	61
2.1.6	Typisierung und Klassifizierung von Objekten.....	63
2.1.7	Zuordnung von Eigenschaften zu Objekten	63
2.1.8	Versionierung mittels IFC-Konstrukten	64
2.1.8.1	GlobalID.....	64
2.1.8.2	IfcOwnerHistory	64
2.1.8.3	Darstellung eines idealtypischen Änderungsworkflows mit Hilfe der IfcOwnerHistory	65
2.1.9	Strukturierung mittels IFC	67
2.1.9.1	Anlagenstrukturen in IFC.....	67
2.1.9.2	Räumliche Gebäudestrukturen und IfcSpace	67
2.1.9.3	Gruppierung von Elementen	71
1.1.1.2	Repräsentation von Heizkreisen im IFC	71
2.1.10	Geometriedarstellung in IFC.....	73
2.1.10.1	Darstellungskontexte	73
2.1.10.2	Repräsentationen.....	73
2.1.10.3	Platzierung.....	74
2.2	Modelica.....	75
2.2.1	Überblick	75
2.2.2	Sprachspezifikation	75
2.2.2.1	Klassen.....	75
2.2.2.2	Komponenten.....	76
2.2.2.3	Konnektoren.....	76
2.2.2.4	Zuordnung von Eigenschaften zu Modelica-Objekten	77
2.2.3	Simulationswerkzeuge.....	77
2.2.4	Model-Layer	79
2.2.5	Ablauf der Simulation	81
2.2.6	Bibliotheken	82
2.2.6.1	MSL.....	82
2.2.6.2	IBPSA	82
2.3	Semantische Technologien	84
2.3.1	Begriffe	84
2.3.2	Wissensgraphen	85
2.3.2.1	Erläuterung des Konzepts	85
2.3.2.2	Syntax	86
2.3.2.3	Inhalte	86
2.3.2.4	Abgrenzung zu alternativen Implementierungen	86
2.3.2.5	Triplestores	87

2.3.2.6	Abfrage und Manipulation von KG mit SPARQL	87
2.3.2.7	Einordnung verwandter Begriffe	87
2.3.3	Ontologien	88
2.3.3.1	Begriff Ontologie	88
2.3.3.2	Basisontologien des W3C	88
2.3.3.3	OWL und Reasoning	89
2.3.3.4	Domänen-Ontologien	90
2.3.3.5	Abgrenzung Ontologie und Wissensgraph	90
2.3.4	Tools zum Handling von Wissensgraphen	91
2.3.4.1	Apache Jena	91
2.3.4.2	Protege	91
2.3.4.3	Rdflib	91
2.3.4.4	Owlready2	91
3	Methodischer Ansatz	93
3.1	Lösungsarchitektur des MO-x-IFC-Toolset	94
3.2	Vorteile durch die Verwendung von Wissensgraphen	95
3.3	Anforderungen an das zu entwickelnde Toolset	97
3.4	Überblick über die MO-x-IFC-Konverter	97
3.5	Schematizer (Konverter 2)	100
3.5.1	Anforderungen / Voraussetzungen der IFC-Files	100
3.5.2	Anforderungen aus dem Handling mit Modelica-Tools	102
3.5.3	Vereinfachungsalgorithmus	103
3.5.4	2D-Platzierung	104
3.5.4.1	Anforderungen	104
3.5.4.2	Grundformen von Graph-Darstellungen	104
3.5.4.3	Gerichtete und ungerichtete Graphen	105
3.5.4.4	Layout-Algorithmen	106
3.6	MoOnt Modelica-Ontologie	107
3.7	Library-Ontologien	110
3.8	SemTran Semantic Translator (Konverter 4)	112
3.8.1	Allgemeines	112
3.8.2	Teilschritte	113
3.8.3	Vorteile des Semantic Reasoning	114
3.8.4	IFC – Modelica – Alignment	114
3.8.4.1	Allgemeines	114
3.8.4.2	Problemklassen bei Zuordnungen	115
3.8.4.3	IFC-AixLib für hydraulische Komponenten	116
3.8.4.4	IFC-IBPSA-Wrapper	118
3.9	TTL2MO (Konverter 5)	119
3.10	MoTTL-Transcriptor (Konverter 7)	120
3.11	ProTran Procedural Translator (Konverter 9)	123
3.12	Voluminizer (Konverter 10)	124
4	Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web	125
4.1	Anforderungen an die Implementierung	126
4.2	Übersicht über die Konverter des MO-x-IFC-Toolsets	126
4.3	Details zu den einzelnen Konvertern MOXIK	127
4.3.1	IFC-seitige Voraussetzungen herstellen (Schritt 1)	127
4.3.1.1	Umwandlung IFC2x3 in IFC4	127

4.3.1.2	Korrektur der Klassen und Typen	128
4.3.1.3	Nachpflege fehlender Verbindungen	128
4.3.1.4	Vergabe von Port-Namen	128
4.3.1.5	Korrektur Containment	129
4.3.1.6	Zuordnung zu Systemen	129
4.3.1.7	Vergeben von Namen	129
4.3.1.8	Verkleinerung der IFC Modelle	129
4.3.1.9	Hilfsfunktionen	130
4.3.2	Schematizer - Abbildung der Vereinfachung in IFC (Konverter 2)	130
4.3.3	IFC2OWL Überführung IFC-STEP in Wissensgraph (Konverter 3)	131
4.3.4	SemTran Semantic Translator (Konverter 4)	132
4.3.4.1	Alignment	132
4.3.4.2	Reasoning	133
4.3.4.3	Enhance_mo	134
4.3.5	TTL2MO Erzeugung von Modelica aus ttl (Konverter 5)	136
4.3.6	MoTTL-Transcriptor (Konverter 7)	138
4.3.7	ProTran Procedural Translator (Konverter 8)	139
4.3.8	RDF2IFC Überführung IFC-OWL in IFC-STEP (Konverter 9)	141
4.3.9	Voluminizer: Erzeugen von Platzhaltergeometrien in IFC (Konverter 10)	141
5	Experimente	143
5.1	UseCase 1: BIM2SIM	144
5.1.1	Überblick	144
5.1.2	Demonstrator Serverkreis	144
5.1.3	Erprobung	146
5.1.3.1	Schritt 1: Vorbereitung	146
5.1.3.2	Konverter 2 Schematizer	151
5.1.3.3	Konverter 3 IFC2RDF	156
5.1.3.4	Konverter 4 Semantic Translator	156
5.1.3.5	Konverter 5 TTL2MO	158
5.1.3.6	Schritt 6: Überführung des Strukturmodells in ein rechenfähiges Simulationsmodell	160
5.1.4	Auswertung	163
5.2	UseCase 2: SIM2BIM	164
5.2.1	Überblick	164
5.2.2	Demonstrator Versuchsstand EAS	165
5.2.3	Erprobung	166
5.2.3.1	MoTTL-Transcriptor	166
5.2.3.2	Procedural Translator	167
5.2.3.3	RDF2IFC	168
5.2.3.4	Voluminizer	169
5.2.4	Auswertung	170
5.3	Vergleichende Auswertung zwischen Semantic und Procedural Translation	170
6	Schlussfolgerungen und kritische Bewertung	173
6.1	Prüfung und Diskussion der Thesen	174
6.2	Limitationen	175
6.2.1	Grenzen der Methodik	175
6.2.2	Grenzen der vorliegenden Implementierung	175
6.3	Mehrwert gegenüber bestehenden Lösungen	175
6.3.1	Übersetzung BIM2SIM	175

6.3.2	Übersetzung SIM2BIM	176
6.4	Ausblick	176
6.4.1	Anknüpfungspunkt Wissensgraph	176
6.4.1.1	Planung Gebäudeautomation.....	176
6.4.1.2	Programmierung der Gebäudeautomation	176
6.4.1.3	Inbetriebnahme und Betrieb von TGA und GA.....	177
6.4.1.4	SIM2BRICK	177
6.4.1.5	Analyse von Simulationsbibliotheken.....	177
6.4.2	Übertragbarkeit auf andere Simulationsmodelle	177
6.4.3	Anwendung für die gekoppelte Simulation	178
6.4.4	Verbesserungsmöglichkeiten in Hinblick auf die Implementierung	178
6.4.5	Auswertung in Hinblick auf Round-Trip.....	179
6.4.6	Einordnung in dDTw-Konzept und Softwarearchitektur von iECO	180
	Verzeichnis verwendeter Software.....	183
	Verzeichnis der zitierten Normen	184
	Verzeichnis wissenschaftlicher Quellen	185
	Anhang	192
	Digitaler Anhang.....	192
	Handling von Wissensgraphen - Größenbegrenzungen.....	192
	Abbildungen Vereinfachung Graph im IFC.....	192

Abbildungsverzeichnis

Abb. 1 Energieflussbild Deutschland 2022 – Angaben in Petajoule (PJ).....	24
Abb. 2 Endenergieverbrauch (Haushalte+GHD) nach Anwendungszweck 2021 – Angaben in Petajoule (PJ)	24
Abb. 3 CO ₂ -Emissionen in Deutschland	25
Abb. 4 Überblick: Teilprozesse und Austauschdokumente im Planungsprozess.....	26
Abb. 5 beispielhafter Ausschnitt aus einem Strangschema	27
Abb. 6 Planungsprozess Netzplanung: Modelle und Dokumente im Planungsprozess.....	28
Abb. 7 beispielhafte Darstellung von Heizkreisen im Schaltschema	29
Abb. 8 Planungsprozess Zentralplanung.....	30
Abb. 9 McLeamy-Kurve.....	32
Abb. 10 Vergleich des Kommunikationsablaufes nach traditioneller Planung und Planung nach der BIM-Methode	42
Abb. 11 Einordnung von Ontologien in Künstliche Intelligenz (KI)	43
Abb. 12 Szenarien der Thermischen/Thermohydraulischen Simulation in der Planungsphase (BauSIM2020)	46
Abb. 13 Übersicht zu den technologischen Kernbestandteilen und den UseCases dieser Arbeit.....	53
Abb. 14 Klassenhierarchie von IfcProduct	60
Abb. 15 Bauteile - definierte Entitäten aus dem Schema IfcProductExtension	61
Abb. 16 definierte Anlagentechnische Elemente aus dem Schema IfcProductExtension	62
Abb. 17 ausgewählte Entitäten in der IFC-Klassenhierarchie	70
Abb. 18 Strukturierung der Räume eines Modell nach unterschiedlichen Kriterien (Screenshot aus KIT-Viewer).....	71
Abb. 19 Subklassen und Superklasse von IfcSystem	71
Abb. 20 Auswahl der Darstellungskontexte im FZK-Viewer	73
Abb. 21 Gegenüberstellung Originalgeometrie und BoundingBox	74
Abb. 22 Funktionsumfang von Modelica-Tools	78
Abb. 23 Benutzeroberfläche des Tools Dymola 2022 mit dem Arbeitsbereich für die Modellerstellung	79
Abb. 24 Diagram Layer eines Modelica-Modells (Ausschnitt).....	80
Abb. 25 Icon und Diagramm Layer eines Beispielmodells (AixLib.Fluid.HeatPumps.HeatPump).....	81
Abb. 26 Prozessschritte und Zwischenergebnisse bei der Bearbeitung eines Executable Modelica-Modells	81
Abb. 27 Packages in der MSL	82
Abb. 28 Package Struktur der IBPSA Libraries	83
Abb. 29 Wissenspyramide, ergänzt nach (Herrmann 2012)	85
Abb. 30 Linked Open Data Cloud, Bildquelle: https://lod-cloud.net/versions/	88
Abb. 31 Wissensgraphen-Stapel für die Domäne Modelica (links) und IFC (rechts) (orange: Domänen-Ontologien, gelb: Instanz-KG)	91
Abb. 32 Status Quo, klassischer und semantischer Lösungsansatz	94
Abb. 33 Zusammenwirken von MO-x-IFC-Konverter (MOXIK) für die UC1 und UC2 zur Überführung zwischen IFC und Modelica	95
Abb. 34 Verknüpfung von Informationen zu Wissen mit Hilfe von Wissensgraphen.....	96
Abb. 35 beteiligte Wissensgraphen für die bidirektionale Übersetzung	98
Abb. 36 Klassendiagramm von wsm, Bildquelle: angepasst Darstellung basierend auf (Wolfram Research 2014)	108
Abb. 37 UML Diagramm MoOnt	109
Abb. 38 Wissensgraphen-Stapel und zugehöriger Stapel von Modelica Bibliotheken	110
Abb. 39 Übersicht zum Translate Schritt zwischen Modelica und IFC.....	113
Abb. 40 Arten von Zuordnungen im Allgemeinen	115
Abb. 41 Screenshot der Wrapper-Bibliothek.....	119
Abb. 42 Screenshot der Parameter der IfcPump-Klasse in der Wrapper-Bibliothek.....	119
Abb. 43 Ablauf auf schematischer und Instanz-Ebene	120

Abb. 44 Gegenüberstellung eines für die Massenermittlung ungeeignet und geeignet aufbereiteten Modelica-Modells.....	123
Abb. 45 Gegenüberstellung „Semantic Translation“ zu „Procedural Tranlation“ für UC4 SIM2BIM BRICK	123
Abb. 46 Komponenten und Wissensgraphen – vorhandene und neue Implementierung	127
Abb. 47 Kälteschema Neubau EAS: Lesbarkeit erst ab Blatthöhe 840 (A0-Querformat und in Überbreite)	130
Abb. 48 Konverter für BIM2SIM.....	144
Abb. 49 IFC-Modell der Architektur des Beispielgebäudes.....	145
Abb. 50 Ausschnitt aus dem Schaltschema des Kaltwassersystems im Gebäude, vollständiges Schema in Abb. 47.....	145
Abb. 51 Konstruktionsmodell des Kaltwassersystems (das betrachtete Teilsystem in rot)	146
Abb. 52 Schemaausschnitt: Verteilerabgang Serverkreis (Quelle: Klemm)	146
Abb. 53 Ursprungsgraph	148
Abb. 54 Komponenten im DG-Modell, die im Originalmodell ohne Verbindung zur Nachbarkomponenten ausgespielt wurden	149
Abb. 55 Zuordnungen aller IfcProduct des fertig aufbereiteten Modell (Screenshot aus simpleBIM)	150
Abb. 56 Ableitung des Funktionsmodells aus dem Konstruktionsmodell am Beispiel des Serverkreises (Darstellung mit matplotlib).....	152
Abb. 57 IfcGroup „replacement_12“ visualisiert in simpleBIM	153
Abb. 58 Gegenüberstellung der 3D-Geometrie	153
Abb. 59 Geometrie des Modells, welches nur noch die berechnungsrelevanten Bauteile enthält	154
Abb. 60 Anzahl der Knoten in den verschiedenen Verarbeitungsstufen des Graphen.....	154
Abb. 61 schematische Darstellung des Serverkreis' im IFC.....	155
Abb. 62 Diagram Layer des erzeugten Modells (Screenshot aus Dymola)	159
Abb. 63 Ausschnitt aus Abb. 51, rot markiert replacement_045 (Flowsegment(163))	160
Abb. 64 globale Parameter des erzeugten Modells (Screenshot aus Dymola)	161
Abb. 65 Parameter einer Modellkomponente (Screenshot aus Dymola)	162
Abb. 66 Diagram Layer des nachbearbeiteten Modells (entspricht XYZ.mo in Abb. 33 und Tab. 29)	163
Abb. 67 Zusammenspiel der Komponenten für den UseCase SIM2BIM	164
Abb. 68 Demonstrator Versuchsstand EAS (entspricht XYZ.mo in Abb. 33 und Tab. 29)	165
Abb. 69 angereichertes 3D-IFC-Modell des HIL-Prüfstands (Screenshots aus dem KIT-Viewer)	170
Abb. 70 Softwarekomponenten für Round-Trip BIM2SIM2BIM	180
Abb. 71 Architektur dDTw auf GAIA-X (Bildquelle: Michael Polter nach (Strnadl u. a. 2024))	181
Abb. 72 Vereinfachung für System RK	193
Abb. 73 Vereinfachung für System Erzeugung	194
Abb. 74 Vereinfachung für System ULK	195
Abb. 75 Vereinfachung für System BKA	196
Abb. 76 Vereinfachung für System RLT	197

Tabellenverzeichnis

Tab. 1 Normen BIM Datenstandards	36
Tab. 2 BIM Modell-, Methoden- und Managementstandards	38
Tab. 3 Übersicht BIM-Tools	40
Tab. 4 IFC-Bibliotheken	41
Tab. 5 Marktübersicht IFC-Viewer	41
Tab. 6 Vergleich von IFC-Viewern anhand ausgewählter Softwarefunktionen	41
Tab. 7 Übersicht Simulatoren	47
Tab. 8 Übersicht Modelica-Bibliotheken für die Gebäudedomäne	48
Tab. 9 wissenschaftliche Veröffentlichung im Bereich BIM2SIM	49
Tab. 10 wissenschaftliche Veröffentlichungen zu IFC2Modelica	50
Tab. 11 Mächtigkeit der 38 Datenschemata in IFC 4.0.2.1	57
Tab. 12 Ausgewählte Types und Entities aus ausgewählten Datenschemata	58
Tab. 13 Attribute von IFCSpace	63
Tab. 14 Attribute von IfcOwnerHistory	65
Tab. 15 Attribute von IfcWall	66
Tab. 16 Attribute von IfcSpatialStructureElement	68
Tab. 17 Beispiel für einfache räumliche Struktur in IFC	68
Tab. 18 Beispiel für komplexe räumliche Struktur in IFC	69
Tab. 19 Möglichkeiten zur Repräsentation von Heizkreisen in IFC	72
Tab. 20 Zuordnung von IfcRepresentationContext, RepresentationIdentifier, RepresentationType und IfcRepresentationItems zu verschiedenen Darstellungsformen	74
Tab. 21 Restriktionen wichtiger Klassen in Modelica	76
Tab. 22 wichtige Modelica Konnektoren	77
Tab. 23 Funktionsumfang von Modelica-Tools im Vergleich	78
Tab. 24 Gegenüberstellung der IBPSA-Bibliotheken	84
Tab. 25 Minimalbeispiel für eine Gebäudeontologie	88
Tab. 26 Basisontologien	89
Tab. 27 Reasoner – ergänzt nach (Noll 2013; „OWL Implementations“ 2020)	89
Tab. 28 OWL Profile (übersetzt nach („OWL 2 Referenz“ 2012; „OWL Referenz“ 2004)	90
Tab. 29 Zwischenstände bei Anwendung von MO-x-IFC für die UC1 und UC2	99
Tab. 30 Übersicht MO-x-IFC-Konverter (MOXIK)	100
Tab. 31 unterstützte Klassen und predefined Types im IFC-AixLib-Mapping	101
Tab. 32 Anforderungen an IFC-Files, die mit MO-x-IFC bearbeitet werden sollen	102
Tab. 33 Graphen nach (Baeldung 2020)	105
Tab. 34 open-source Layout-Algorithmen für Graphen	106
Tab. 35 Gegenüberstellung Ontologien im Bereich IFC und Modelica	110
Tab. 36 Kennzahlen der entwickelten Ontologien	111
Tab. 37 Problemklassen in Bezug auf Übersetzung	116
Tab. 38 Mapping für hydraulische Komponenten	117
Tab. 39 Mapping-Tabelle IFC Modelica für Komponenten von Lüftungsanlagen	118
Tab. 40 Übersicht MO-x-IFC-Konverter (MOXIK)	126
Tab. 41 Skalierung und Platzierung der verschiedenen Layouts	131
Tab. 42 Erläuterung der Klassen im Alignment	133
Tab. 43 Parameter der verwendeten AixLib Komponenten	138
Tab. 44 zusammengehörige Versionen von MoOnt und MoTTL-Transcriptor	139
Tab. 45 Platzhalter-Geometrien	142

Tab. 46 Bezeichnung der Zustände im Experiment zu UC1 BIM2SIM	144
Tab. 47 Aufbereitungsschritte für das Beispielfile	149
Tab. 48 Effekt der Größenreduzierung der IFC-STEP-Dateien am Beispiel EAS KLT (für alle Teilsysteme)	151
Tab. 49 Anzahl der Knoten in den verschiedenen Verarbeitungsstufen des Graphen	155
Tab. 50 Kennzahlen beteiligte Wissensgraphen (Beispiel Serverkreis).....	156
Tab. 51 Laufzeiten Reasoning	157
Tab. 52 Nachbearbeitung der MO-Datei.....	160
Tab. 53 Bezeichnung der Zustände im Experiment zu UC1 BIM2SIM.....	164
Tab. 54 FLOSS-Software	183
Tab. 55 kommerzielle Software	183
Tab. 56 zitierte Normen	184
Tab. 57 Zusammenstellung der zur Arbeit gehörigen Quelltexte, Beispieldateien und Binaries	192
Tab. 58 Handling großer Textdateien – Grenzen der Werkzeuge.....	192

Verzeichnis von Quelltext- und sonstigen Dateiausschnitten

Lis. 1 Ausschnitt aus der DATA-Sektion (DATA section) einer IFC-STEP-Datei.....	59
Lis. 2 Drei Varianten einen Raum innerhalb von IFC zu klassifizieren	63
Lis. 3 IfcChangeActionEnum	65
Lis. 4 Beispielhafte Entitäten für die Erstellung einer OwnerHistory	65
Lis. 5 Triple als Pseudocode	85
Lis. 6 Beispiele für Instanz- (orange) und Schemawissen (blau) als Pseudocode	85
Lis. 7 gültiges RDF-Tripel serialisiert als RDF/XML (entspricht Tripel 5 aus Lis. 6)	86
Lis. 8 RDF-Tripel serialisiert als Turtle-Syntax (entspricht Tripel 5 aus Lis. 6)	86
Lis. 9 Beispieltriple aus A-Box, T-Box und der Verbindung von A- und T-Box in Turtle-Syntax Orange: Instanzwissen (A-Box), blau: Schemawissen (T-Box), pink: Verbindung.....	86
Lis. 10 Beispiel für eine SPARQL query	87
Lis. 11 Beispiel für ein PropertyChainAxiom	87
Lis. 12 Beispiel Reasoning	89
Lis. 13 Klassendefinition der Wrapper-Komponente für IfcPump	119
Lis. 14 Ausschnitt aus der Transkription der AixLib in einen KG (grün: statement aus MoOnt, blau: statements aus AixLib)	121
Lis. 15 Vergleich eines Ausschnitts aus dem Carnot_y-Wärmepumpenmodell (Bibliothekskomponente) als natives Modelica file (oben) und als Mo-KG (unten) turtle syntax.....	121
Lis. 16 Vergleich eines Ausschnitts aus dem heatPumpPlant (Modelica Executable) als natives Modelica file (oben) und als Mo-KG (unten) turtle syntax	122
Lis. 17 Beispiel Header-Abschnitt einer IFC-STEP-Datei	128
Lis. 18 Ausschnitt aus Beispiel-IFC B25.ifc (STEP-Syntax)	131
Lis. 19 Ausschnitt aus Beispiel IFC als KG B25.ttl (Turtle-Syntax), dieser umfasst einen Teil der Informationen in Lis. 18	132
Lis. 20 Aufruf der IFCtoRDF Kommandozeilenanwendung	132
Lis. 21 Ausschnitt aus Alignment zur Zuordnung von Umwälzpumpen in IFC und AixLib (ttl-Syntax)	133
Lis. 22 Ausgewählte Triple aus XYZ_ifc.ttl gemäß Tab. 46	133
Lis. 23 Semantic Reasoning	134
Lis. 24 SPARQL query für die Ergänzung von Informationen im Mo-KG	135
Lis. 25 Codeausschnitt zur Übertragung der Positionsinformationen aus IFC-KG in Mo-KG	135
Lis. 26 Codeausschnitt zur Übertragung der connections aus IFC-KG in Mo-KG	136
Lis. 27 TTL2MO Teil 1: Komponenten finden und erzeugen (Parametrierung in orange)	137
Lis. 28 TTL2MO Teil 2: Finden und Erzeugen von Verbindungen	138
Lis. 29 ProTran: wesentliche Bestandteile (nur ausschnittsweise wiedergegeben)	139
Lis. 30 ProTran: Funktionen zur Erzeugung der Klassenzuordnungen (Ersatz für das Reasoning)	140
Lis. 31 Ausschnitt aus der Library-Ontologie einer benutzerdefinierten Modelica-Bibliothek	141
Lis. 32 Aufruf der IfcOWL-to-IfcSTEP Kommandozeilenanwendung	141
Lis. 33 Aufruf des Konverter 3 IFC2RDF	156
Lis. 34 Ausschnitt aus geschlussfolgerten Triple KG XYZ_classes_mo.ttl in Turtle-Syntax	157
Lis. 35 Ausschnitt aus Beispiel moont als KG XYZ_mo.ttl (Turtle-Syntax), farblich hervorgehoben sind die Ergänzungen zu Lis. 34	158
Lis. 36 Ausschnitt aus HIL_mo.ttl (Imports farbig hervorgehoben) (entspricht XYZ_mo.ttl in Abb. 33 und Tab. 29)	166
Lis. 37 Auszug aus der HIL_ifc.ttl (entspricht XYZ_ifc.ttl in Abb. 33 und Tab. 29).....	168
Lis. 38 Ausschnitt aus der HIL_F.ifc (entspricht XYZ_F.ifc in Abb. 33 und Tab. 29).....	169

Abkürzungsverzeichnis

Als Begriff verwendete Abkürzungen finden sich im Glossar

Abkürzung	Bedeutung
AIA	Auftraggeber-Informations-Anforderungen
Aix	Namespacékürzel
AixLib	Eigenname für die Modelica Library
AS	Anlagensimulation
BAP	BIM-Abwicklungsplan
BOT	Building Topology Ontology
BRICK	Ontologie
bS	buildingSMART (Verein)
CDE	Common Data Environment
CEN	Comité Européen de Normalisation, Europäisches Komitee für Normung
CFD	Computational Fluid Dynamics
DAE	Differential algebraic and discrete equations
DIN	Deutschen Institut für Normung
ESIM	Ontologie
FLOSS	Free/Libre and Open Source Software
GA	Gebäudeautomation
GHD	Gewerbe, Handel, Dienstleistungen
GS	Gebäudesimulation
HLK	Heizung, Lüftung, Kälte
HOAI	Verordnung über die Honorare für Architekten- und Ingenieurleistungen
i.d.R.	In der Regel
IFC	Industrial Foundation Classes
IFC-KG	IFC-Knowledge Graph
insb.	Insbesondere
ISO	International Standardisation Organisation
KB	Knowledge Base
KG	Knowledge Graph
MBL	Modelica Buildings Library
MDLG	Modelica-Diagram-Layer-Größeneinheiten
MLS	Modelica Language Specification
mo	Dateiendung von Modelica-Dateien
Mo-KG	Modelica-Knowledge Graph
MoC	Model of Computation
MO-x-IFC	Gewähltes Kürzel für den entwickelten Workflow
MOXIK	MO-x-IFC-Komponente, entspricht PS
MSL	Modelica Standard Library
owl, OWL	Web Ontology Language
PPP	Pre- und Postprocessing
PS	Prozessschritt
RDF	Resource Description Framework
RDFS rdfs	Namespacékürzel, Resource Description Framework Schema
red	Index für „reduziert“
sog.	sogenannt
STEP	STandard for the Exchange of Product model data
SWT	Semantic Web Technologies
TGA	Technische Gebäudeausrüstung
TTL, ttl	Turtle
UC	UseCase
VDI	Verein deutscher Ingenieur
W3C	World Wide Web Consortium
WG	Wissensgraph, auch KG
xyz	Platzhalter für Bezeichnung des Beispielmodells
ZS	Zustand, Zwischenschritt, Zwischenstand

Glossar

Vorbemerkung zur sprachlichen Darstellung:

- Informationen aus dem IFC-Schema im Text: In der Regel werden keine Übersetzungen vorgenommen, sondern die Fachbegriffe aus dem Schema verwendet. i.d.R. werden dafür auch keine Übersetzungen angegeben, da dem Leser entsprechend Englischkenntnisse unterstellt werden und die Lesbarkeit somit besser wird. Die Begriffe werden *kursiv und mit einer grauen Unterlegung* hervorgehoben.
- Um die Zuordnung zu den englischsprachigen Referenzdokumenten zu erleichtern, werden im Folgenden die *englischen Begriffe* und *Eigennamen* innerhalb des deutschen Textes verwendet. Im Sinne der Lesbarkeit wird auf die Verwendung von Anführungszeichen verzichtet und stattdessen *Kursivstellung* genutzt.

Begriff (Kontext)	Erläuterung und Verweise innerhalb dieser Tabelle	Verweis auf Einführungstexte (Abschnitt bzw. Seite)
A-Box	spezielle Art von →Wissensgraph, die →Instanzwissen enthält Synonym zu →Instanz-KG	2.3.3.5
Alignment	Ist eine spezielle Realisierung eines Mappings in Form eines Wissensgraphen Abgrenzung: Mapping, Zuordnung	1.2.3.2
Anlagengraph	Struktur aus Komponenten einer Anlage und den zugehörigen Verbindungen	S. 67
Anlagenschema	Synonym zu →Schema, „Überbegriff zu →Strang- und →Schaltschema“	-
Annotation	Zuordnung einer → Eigenschaft zu einer Klasse in Modelica Abgrenzung: → parameter, → stringComment, → documentation string	2.2.2.4 S. 77
Attribut	Zuordnung einer → Eigenschaft zu einer Klasse im IFC-Standard Abgrenzung: → Property	2.1.7 S. 63
Autorentool		1.2.2.1
Berechnungsmodell	Spezialform eines →Funktionsmodells	31
berechnungsrelevant	Menge der Informationen, die für eine Berechnung nötig sind Im Falle einer Simulationsberechnung bedeutet berechnungsrelevant = →simulationsrelevant (beide Begriffe werden daher in dieser Arbeit synonym verwendet) Abgrenzung: nicht-berechnungsrelevant	-
DataTypeProperty	<i>Art eines Objekte in RDF</i> <i>Synonym zu Literal</i>	2.3.2.1
documentation string	Zuordnung einer → Eigenschaft zu einer Klasse in Modelica Begrifflichkeit aus MLS Synonym: →stringComment Abgrenzung: → parameter, → annotation	2.2.2.4 S. 77
Domäne	In dieser Arbeit wird unterschieden zwischen - den beiden Fach-Domänen Modelica und IFC - sowie der semantischen Domäne im Gegensatz zur Domäne der nativen Dateiformate	
Eigenschaft	Überbegriff zu → Attribut und → Property bzw. Quantity in der IFC-Domäne, sowie zu → Parameter, → annotation, → stringComment, → documentation string in der Modelica-Domäne	2.1.7 S. 63
Eingabeparameter	Für CLI, in dem Zusammenhang wird bewusst nicht der Begriff →Parameter verwendet	-
Executable	→Modelica-Executable	
Funktionsmodell	Wichtiges Modell in der Planung, enthält alle →strukturrelevanten Bauteile Abgrenzung: →Konstruktionsmodell	30
Gebäudeautomation	Überbegriff zu Raum- und Anlagenautomation	-
GlobalID	Globally Unique Identifier für Ifc-Root-Objekte, →Identifikator in IFC-Modellen	2.1.4.1
Heizkreis	Gruppe der Komponenten hinter einem Verteilerabgang →Kühlkreis	28
IFC-Funktionsmodell	Repräsentation eines Funktionsmodells im Format IFC	Tab. 29
IFC-KG	→Zwischenstand des →MO-x-IFC-Workflows	Tab. 29
IFC-Schaltschema	Zweidimensionale Repräsentation des Funktionsmodells in IFC	S. 29
IFC-Viewer	Bestimmte Art eines Softwarewerkzeugs für die Handhabung von IFC-Modellen	1.2.2.1, 1.2.2.3
IfcProperty	Synonym zu →Property	-
Identifikator	Eineindeutig kennzeichnendes Merkmal in einer bestimmten Domäne z.B. →identifier und →GlobalID	-
identifier	→Identifikator in Modelica-Modellen	2.2.2.2

Begriff (Kontext)	Erläuterung und Verweise innerhalb dieser Tabelle	Verweis auf Einführungstexte (Abschnitt bzw. Seite)
Information	Abgrenzung zu →Wissen	2.3.1
Instanz	Begriff wird in der Arbeit auf 3 Arten benutzt, wenn Verwechslungsgefahr besteht, werden konkretere Begriffe verwendet 1. Liste von Instanzen in STEP-Files →STEP-Instanz 2. Instanzen von Klassen in Modelica bilden →Komponenten in Modelica-Modellen 3. bei der Unterscheidung zwischen den Wissensarten →Instanzwissen (A-Box) und →Schemawissen (T-Box) 4. <i>individuals</i> in einem Wissensgraph (rdf:type), im Gegensatz zu Klassen in einem Wissensgraph (rdf:subClassOf, rdf:equivalentClass)	-
Instanz-Objekt	Art von Objekten in der IFC-Domäne Gegenteil von →Type-Objekt	S. 57
Instanz-KG Instanz-Graph	Instanzwissen in Form eines →Wissensgraph Synonym zu →A-Box Abgrenzung: →Ontologie	-
Instanzwissen	→Wissen über konkrete Zusammenhänge der realen Welt Kann auf vielerlei Art abgespeichert werden, z.B. als Foto, Messdaten in Excel, IFC-Modell, Modelica-Modell Abgrenzung: →Schemawissen	-
Kind-Klasse	Bestandteil einer Vererbungshierarchie, in Abgrenzung zur Eltern-Klasse	-
Knowledge Graph (KG)	Wird i.d.R. nicht verwendet, die Abkürzung KG hingegen schon Synonym: →Wissensgraph, →RDF-Wissensgraph	2.3.2.1
Knowledge-Base (KB)	Zusammenstellung mehrerer →Wissensgraphen Besteht aus →A-Box und →T-Box	2.3.3.5
Komponente (Modelica-Modell)	Im Kontext eines Modelica-Modells: Instanzen von Klassen in Modelica bilden Komponenten in Modelica-Modellen	2.2.2 S. 75
Konstruktionsmodell	Neben dem →Funktionsmodell wichtiges Modell in der Planung	30
Konverter	Bestandteile des →MO-x-IFC-Toolset	3.1
Kühlkreis	Analoges Konzept zu →Heizkreis in der Kühlung von Gebäuden	S. 28
Library	Softwarebibliothek In dieser Arbeit i.d.R. verwendet für →Modelica-Bibliotheken Ausnahmsweise auch für Python- oder Java-Bibliotheken	-
Mapping	Im Text Hauptsächlich verwendet für den umgangssprachlichen Begriff „Zuordnung“ Abgrenzung: Alignment Synonym: Zuordnung	-
Mapping-Tabelle	Informationen bzgl. der Zuordnung von Entitäten in zwei Domänen	-
maschinenlesbar	Für die automatisierte Weiterverarbeitung gedachte Informationen, im Gegensatz zu <i>menschlesenbarem</i> Informationen	-
Mo-KG	→Zwischenstand des →MO-x-IFC-Workflows	Tab. 29
MO-x-IFC-Toolset	Überbegriff über die in dieser Arbeit entwickelten Konverter	3.1
MO-x-IFC-Workflow	Arbeitsschritte, die mit dem → MO-x-IFC -Toolset unterstützt werden sollen	3.1
Modelica-Bibliothek	Auch →Library	2.2.6
Modelica-Executable	Ausführbarer Simulationsmodell in Modelica	2.2.2.1
Objekt	Bestandteil eines →Anlagengraph im IFC-Modell (→Konstruktions- oder →Funktionsmodell) Hat ggf. ein Pendant im →Simulationsmodell	-
Ontologie	spezielle Art von →Wissensgraph, die →Schemawissen enthält Schemawissen in Form eines →Wissensgraph (vorher bezeichne ich es nicht als Ontologie, obwohl der Begriff das hergeben <u>würde</u>) Abgrenzung: →Instanz-KG Synonym →T-Box	2.3.3.1
Parameter	Zuordnung einer → Eigenschaft zu einer Klasse in Modelica Abgrenzung: → annotation, → stringComment, → documentation string, → Eingabeparameter	2.2.2.4 S. 77
predefinedType	Attribut, welches viele Klassen im IFC-Schema tragen	2.1.4.2
Procedural Translation	Alternative zu →Semantic Translation	3.1
Procedural Translator	→Konverter im →MO-x-IFC-Toolset, welcher →Procedural Translation realisiert	3.1

Verzeichnisse

Begriff (Kontext)	Erläuterung und Verweise innerhalb dieser Tabelle	Verweis auf Einführungstexte (Abschnitt bzw. Seite)
Property	Zuordnung einer → Eigenschaft zu einem Objekt in IFC mit Hilfe der IfcRelDefinesByProperties-Relation Im Kontext OWL gibt es auch viele Begrifflichkeiten die „property“ enthalten (z.B. →PropertyChain, →DataTypeProperty, ObjectProperty), jedoch nicht das allein stehende Wort „Property“. An Textstellen wo Verwechslungsgefahr bestünde wird daher in Abgrenzung zur OWL Domäne →“IfcProperty“ statt „Property“ verwendet Abgrenzung: → Attribut Synonym: → IfcProperty	2.1.7 S. 63
PropertyChain	Beliebig lange Ketten von Eigenschaftszuordnungen in →Wissensgraphen	2.3.2.6
RDF-Wissensgraph	In dieser Arbeit synonym verwendet zu →Wissensgraph, da andere Wissensgraphen nicht behandelt werden	2.3.2.4
Reasoner	Software zum Schlussfolgern basierend auf OWL-Konstrukten	2.3.3.3
Reasoning	Deutsch: Schlussfolgern Begriff wird hier bewusst als englischer Term verwendet, um Verwechslungen mit dem umgangssprachlichen Wort auszuschließen	2.3.3.3
Reasoning-Engine	Nicht verwendet, → Reasoner	-
Ressourcenentitäten	Datenstrukturen aus dem resource layer in IFC, können nicht eigenständig existieren, sondern nur, wenn sie von anderen Entitäten referenziert werden	S. 56
Schalschema	Austauschdokument zur Darstellung der Zentralplanung, wird ergänzt durch das →Strangschemata derselben Anlage	S. 29
Schema	Synonym zu →Anlagenschema, Überbegriff zu →Strang- und →Schalschema	-
schemarelevant	Menge der Informationen, die üblicherweise in einem Schema dargestellt werden. In Bezug auf die dargestellten Bauteile, betrifft es jene die Änderungen am Fluidstrom hervorrufen, die über einen konstanten Druckverlust hinausgehen, sowie sicherheitsrelevante und weitere Bauteile →Simulationsrelevant, →berechnungsrelevant, →strukturrelevant	S. 30
Schemawissen	→Wissen über allgemeine Zusammenhänge der realen Welt Kann auf vielerlei Art existieren: z.B. das Wissen, dass Fenster aus Rahmen und Glas bestehen (ist auch so formuliert im IFC-Schema) Abgrenzung: →Instanzwissen	-
Semantic Reasoning	Synonym zu →Reasoning	2.3.3.3
Semantic Translation	Alternative zu →Procedural Translation	3.1
Semantic Translator	→Konverter im →MO-x-IFC-Toolset, welcher →Semantic Translation realisiert	3.1
shape	Kombination aus Konnektoren und Parameter einer Modelica-Klasse	76
Simulationskern	Synonym: →Simulator	1.3.2
Simulationsmodell	Spezialform eines →Berechnungsmodells	31
simulationsrelevant	Menge der Informationen, die für eine Simulation nötig sind Simulation ist dabei eine Form der Berechnung, der Begriff wird daher in dieser Arbeit synonym zu →berechnungsrelevant verwendet	-
Simulationsumgebung	Enthält mehrere →Simulationswerkzeuge	2.2.3
Simulationswerkzeug	Überbegriff zu →Simulator, Benutzerschnittstelle, Modellbibliotheken	1.3.2, 2.2.3
Simulator	Das Simulationswerkzeug, welches die Simulation durchführt (der Ablauf einer Simulation wird in Abschnitt 2.2.5 beschrieben Synonym: Simulationskern	1.3.2
Stand	Begriff auf dem Kontext „Planungsprozess“ Arbeitsstand eines Modell Üblich ist auch die Verwendung der Begriffe „Version“ oder „data drop“	1.2
STEP-Instanz	Verwendung, wenn Abgrenzung zu anderen Nutzungen des Begriffs → Instanz nötig	2.1.2 S. 56
Strangschemata	Austauschdokument zur Darstellung der Netzplanung, wird ergänzt durch das →Schalschemata derselben Anlage	S. 29
stringComment	Zuordnung einer → Eigenschaft zu einer Klasse in Modelica Begrifflichkeit aus moont Synonym: → documentation string Abgrenzung: → parameter, → annotation	2.2.2.4 S. 77
Strukturmodell	→Zwischenstand des →MO-x-IFC-Workflows	Tab. 29
strukturrelevant	Schnittmenge der →schemarelevanten Informationen mit den →berechnungs- bzw. →simulationsrelevanten Informationen	S. 31
Submodell	Begriff aus der Modelica-Domäne zur Beschreibung der Verschachtelung von Modellen, Submodelle sind →Komponenten von →Supermodellen	-
Supermodell	Abgrenzung zu →Submodell	-
T-Box	Synonym zu →Ontologie	2.3.3.1
Toolset	→MO-x-IFC-Toolset	-

Begriff (Kontext)	Erläuterung und Verweise innerhalb dieser Tabelle	Verweis auf Einführungstexte (Abschnitt bzw. Seite)
Transkription	Prozess der Übertragung von nativen Datenformaten in Wissensgraph Zugehöriges Werkzeug heißt allgemein →Transkriptor	3.1
Transkriptor	Allgemeines Bezeichnung eines Werkzeugs zur →Transkription Konkrete Tools in dieser Arbeit <i>MoTTL-Transcriptor</i> <i>RDF2IFC</i> <i>IFC2RDF</i> <i>MoTTL-Rescriptor</i>	3.1
Translation	Prozessschritt im →MO-x-IFC-Toolset, kann als →Semantic oder →Procedural Translation realisiert werden	100
Translator	Oberbegriff über →Konverter im →MO-x-IFC-Toolset, welche die →Übersetzung zwischen zwei Domänen realisieren	3.1
Type-Objekt	Art von Objekten in der IFC-Domäne Gegenteil von →Instanz-Objekt	S. 57
type	Bezeichnung für das Konzept <i>Klasse</i> in Modelica	2.2.2.1
Übersetzung	→Übertragung zwischen in zwei Domänen	-
Übertragung	Oberbegriff zu →Transkription und →Translation	-
Updater	Softwarekomponente zum zusammenführen von zwei Modellständen	6.4.5
UseCase	Anwendungsfall für den Einsatz des MO-x-IFC-Toolset	-
Wissen	Wissen entsteht durch Verknüpfung von →Informationen Es wird unterschieden in →Schemawissen und →Instanzwissen	2.3.1
Wissensgraphen (KG)	Sehr allgemeines Datenmodell/Art der Datenspeicherung Wird in dieser Arbeit immer in Form von Files gespeichert Zugehöriger Standard: RDF, verwendete Serialisierung TTL Inhalt eines KG kann →Schemawissen (dann wird der KG als →Ontologie bezeichnet) oder →Instanzwissen sein (dann wird der KG als →Instanz-KG bezeichnet) Mehrere KG bilden eine → Knowledge-Base Synonym: →RDF-Wissensgraph	2.3.2.1
Wrapper-Bibliothek	Hilfskonstrukt zur eindeutigen Zuordnung zwischen Modelica und IFC, erstellt in der Modelica-Domäne	3.8.4.4
Zuordnung	Synonym zu → Mapping	-

1 Motivation, Stand von Wissenschaft und Technik

Im diesem Kapitel wird zunächst hergeleitet, dass Gebäude einen wesentlichen Hebel für die Erreichung der Klimaziele bieten. Ein entscheidender Aspekt ist dabei eine energieeffiziente Anlagentechnik. Diese kann insbesondere durch den Einsatz von Simulationsverfahren in der Planungsphase erreicht werden. Es wird aufgezeigt, wie der Planungsprozess klassischerweise abläuft und wie sich Simulationsverfahren darin einordnen. Als wesentliche Hürde dafür erweist sich der Aufwand für die Modellerstellung. Diese kann durch die Nutzung von Building Information Management (BIM) abgemindert werden.

Anschließend wird der Stand von Wissenschaft und Technik im Bereich BIM dargestellt. Es wird ein kurzer Überblick über die bestehende Normenlandschaft und die marktverfügbarsten Tools gegeben, sowie der Gegenstand aktueller Forschungsarbeiten zusammengefasst. Für die weiteren Arbeiten wird das Format IFC ausgewählt, da es das open-BIM-Prinzip erfüllt, bereits weite Anwendung in der Praxis findet und sehr ausdrucksstark ist.

Im nächsten Abschnitt wird der Stand von Wissenschaft und Technik im Bereich Gebäude- und Anlagensimulation zusammengefasst. Auch hier wird ein kurzer Überblick über die bestehende Normenlandschaft und die marktverfügbarsten Tools gegeben, sowie der Gegenstand aktueller Forschungsarbeiten zusammengefasst. Für die weiteren Arbeiten wird die Modellierungssprache Modelica ausgewählt, da dafür leistungsfähige Bibliotheken, Simulatoren und Benutzerschnittstellen als open-source-Software verfügbar sind.

Wesentlich für den Einsatz der Anlagensimulation in der Planungsphase ist das Zusammenwirken von BIM und Simulation. Dies ist Gegenstand der Forschung, aber noch nicht „Stand der Technik“. Es wird ein Überblick über die Forschungsarbeiten in diesem Bereich gegeben. Semantische Technologien erscheinen vielversprechend als Lösungsansatz für das Zusammenwirken von BIM und Simulation, daher wird der Stand der Forschung im Bereich der Ontologien für Modelica-Modelle ebenfalls dargestellt.

Im letzten Abschnitt wird die Aufgabenstellung für diese Arbeit abgeleitet und zwei wesentliche praktisch relevante Use-Cases definiert:

- BIM2SIM: Erstellung eines Simulationsmodells aus einem BIM-Modell
- SIM2BIM: Erstellung eines BIM-Modells basierend auf einem Simulationsmodell

Es werden die zu untersuchenden Thesen aufgestellt und ein Überblick über den Aufbau der weiteren Arbeit gegeben.

1.1 Motivation

1.1.1 Gebäude im Kontext des Klimawandels

Fast die Hälfte der Endenergie (43%) in Deutschland wird in den Sektoren Haushalte und Gewerbe, Handel, Dienstleistungen (GHD) verbraucht (Abb. 1). Deren Verbrauchsverhalten ist, wie Abb. 2 zeigt, stark vom Wärmebedarf dominiert. Nimmt man die Verbräuche für Kühlung und Beleuchtung hinzu, erkennt man den großen Einfluss der technischen Gebäudeausrüstung (TGA) auf den Energieverbrauch der Industrienation Deutschland.

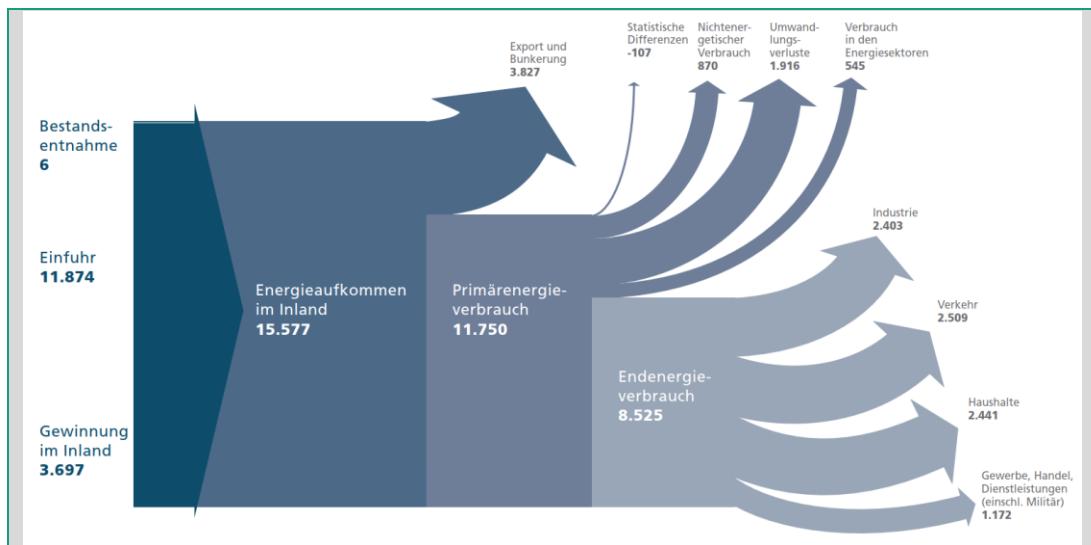


Abb. 1 Energieflussbild Deutschland 2022 – Angaben in Petajoule (PJ)

Bildquelle (AGEB 2023)

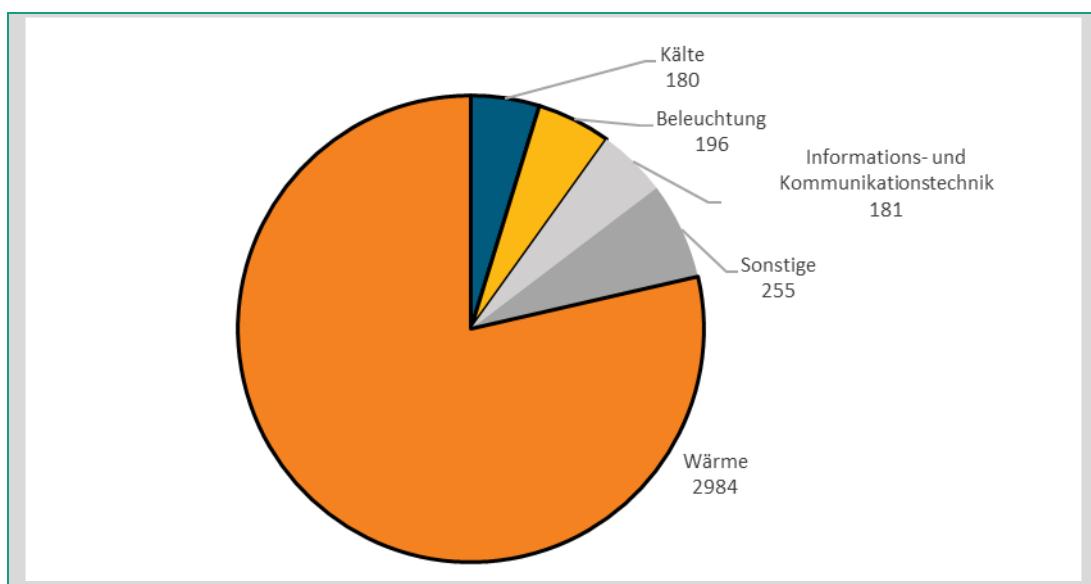


Abb. 2 Endenergieverbrauch (Haushalte+GHD) nach Anwendungszweck 2021 – Angaben in Petajoule (PJ)
89% des Endenergieverbrauchs werden von TGA-Anlagen verursacht

eigene Darstellung nach (AGEB 2022)

Vor dem Hintergrund des Klimawandels ist es notwendig, die CO₂-Emissionen zu senken. Abb. 3 zeigt die CO₂-Emissionen in Deutschland 2020 bzw. 2021. Man erkennt den erheblichen Anteil,

welcher direkt (Gebäude) und indirekt (Energiewirtschaft, Industrie²) durch Gebäude verursacht wird. Um zu einer signifikanten Senkung der CO₂-Emissionen zu kommen, bieten energieeffizient betriebene Gebäude daher neben Industrie und Mobilität den entscheidenden Hebel – knapp die Hälfte der jährlich in Deutschland ausgestoßenen CO₂-Äquivalente werden durch Gebäude verursacht, der überwiegende Teil davon in der Betriebsphase.



Abb. 3 CO₂-Emissionen in Deutschland

Bildquelle: (Braune u. a. 2022)

Wesentlichen Einfluss auf den Energieverbrauch von Gebäuden hat, neben der Gebäudehülle und dem Nutzerverhalten, das in der Planungsphase festgelegte Anlagenkonzept für die Wärme- bzw. Kälteversorgung, sowie die Umsetzung der geplanten Konzepte in der Betriebsphase. Aufgrund des in der Literatur häufig beschriebenen „Performance Gap“ (van Dronkelaar u. a. 2016; Imam, Coley, und Walker 2017; Cozza u. a. 2021) ist es wichtig, Planung und Umsetzung von Konzepten separat zu bewerten. Diese Arbeit entwickelt Werkzeuge, um das Erstellen und Umsetzen energieeffizienter Anlagenkonzepte zu vereinfachen.

1.1.2 Planungsprozess der technischen Gebäudeausrüstung

Beim Erstellen und Umsetzen energieeffizienter Anlagenkonzepte sind zahlreiche Akteure beteiligt: Eigentümer, Nutzer, Facility Manager, Planer, Baufirmen, Politik und Behörden, sowie die Finanzbranche. In den Lebenszyklusphasen Planung, Errichtung, Inbetriebnahme und Betrieb werden jeweils wichtige Weichen gestellt, die den Endenergieverbrauch eines Gebäudes für die thermische Konditionierung beeinflussen.

Die wesentlichen Entscheidungen zum Anlagenkonzept werden in der Vorplanung vom Fachplaner für technische Gebäudeausrüstung (TGA) getroffen. Dieser agiert dabei in enger Abstimmung mit dem Architekten als Entwurfsverfasser und den anderen beteiligten Fachplanern insb. für Bauphysik, Elektrotechnik und Gebäudeautomation. Die Vorplanung „stellt die zur Aufgabenstellung passende, technisch und wirtschaftlich günstigste Lösung als Grundlage für die weitere Bearbeitung und die zugehörigen Konzepte dar“ (VDI 6026-1). Die Vorplanung ordnet sich als zweite in die neun in der HOAI (BMJ 2013) definierten Leistungsphasen ein:

² Im Sektor Industrie wird die „graue Energie“ für die Herstellung der Baumaterialien bilanziert.

1 Motivation, Stand von Wissenschaft und Technik

1.1 Motivation

1.1.2 Planungsprozess der technischen Gebäudeausrüstung

1. Grundlagenermittlung
2. Vorplanung
3. Entwurfsplanung
4. Genehmigungsplanung
5. Ausführungsplanung
6. Vorbereitung der Vergabe
7. Mitwirkung bei der Vergabe
8. Bauüberwachung
9. Objektbetreuung

Abb. 4 zeigt den Ablauf der Vorplanung für die Gewerke Heizung und Kühlung, sowie der zugehörige Gebäudeautomation (GA). In der blauen Zeile sind die wesentlichen Austauschdokumente dargestellt. Hier ist bewusst von „Dokumenten“ die Rede, da es gemäß HOAI bzw. VDI 6026 (Dokumentation TGA) i.d.R. eine Verpflichtung gibt, diese zu erstellen. Sie haben sich dementsprechend als geeignete Schnittstellen zwischen verschiedenen Prozessbeteiligten etabliert. Der Planungsprozess kann unterteilt werden in die Planung des Netzes (der Begriff „Netz“ bezieht sich im Folgenden immer auf Wärme- bzw. Kälteverteilnetze), der Zentrale und der Gebäudeautomation (GA). Die „Zentrale“ enthält neben den Erzeugungsanlagen i.d.R. weitere Verbraucheranlagen, z.B. Trinkwarmwasserbereitung, Prozesskühlung, Lüftungsanlagen. Diese Verbraucheranlagen verursachen oftmals einen Großteil der notwendigen Anschlussleistung und des Energiebedarfs, die Verbraucher im Netz (Raumheizung/-kühlung) sind demgegenüber oft nachgeordnet. Die Charakteristik der **Netzverbraucher** und **Verbraucheranlagen** ist oftmals sehr unterschiedlich, weshalb beide **Verbrauchergruppen** in der Planung separat behandelt werden. An die Planung von Netz und Zentrale schließt sich die Planung der Gebäudeautomation an.

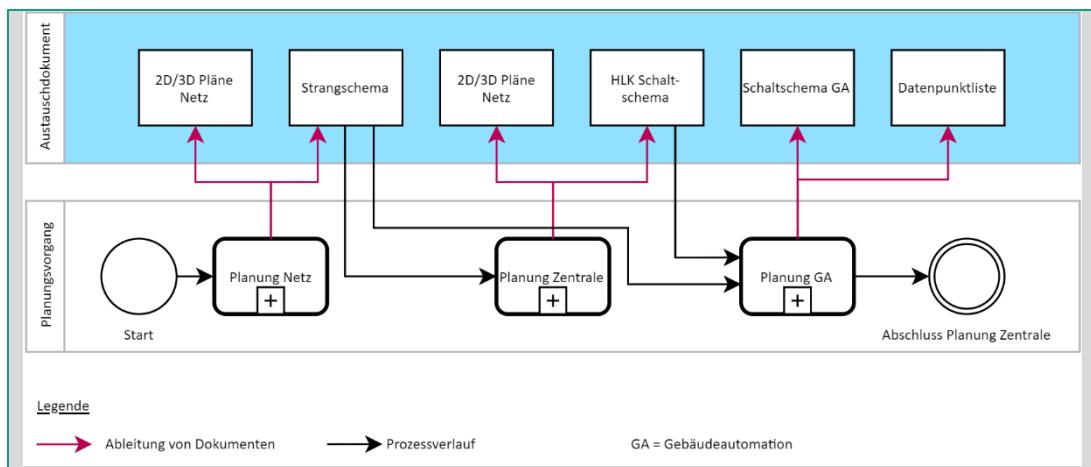


Abb. 4 Überblick: Teilprozesse und Austauschdokumente im Planungsprozess

Die Planung beginnt i.d.R. mit der Betrachtung der Räume (**Netzplanung**) wie in Abb. 6 dargestellt. Zunächst werden Anforderungen durch Bauherrn und Architekten festgelegt. Das übliche Austauschformat dazu ist das **Raumbuch**. Dieses wird durch die Fachplanung zu einem Raummodell ergänzt, z.B. durch Ergänzung der raumweisen thermischen Lasten und die Festlegung der heizungs- und kälteseitigen Übergabeelemente. Aus dem Raummodell gemäß VDI 6070 kann ein „Raumbuch TGA“ abgeleitet werden. Anschließend werden die groben Versorgungsstrukturen im Funktionsmodell Netz festgelegt, es erfolgt eine Zuordnung der einzelnen Verbraucher (z.B. Heizkörper) zu sog. Heiz- bzw. Kühlkreisen. Danach erfolgt im Zuge der Netzkonstruktion die Anreicherung der geometrischen Informationen im Konstruktionsmodell,

welche in einem iterativen Verfahren in das Funktionsmodell rückgeführt werden muss, soweit es für die Berechnung relevant ist. Rohrführung und -dimensionierung werden iterativ solange angepasst, bis die geometrischen Anforderungen (z.B. aus Kollisionsprüfung) und die technischen Anforderungen (z.B. zulässige Druckverluste) eingehalten sind. Aus der finalen Version des Funktionsmodells kann dann das sog. **Strangschema** als Austauschdokument abgeleitet werden. Dieses enthält nun die raumseitigen Anforderungen an die Erzeugungsanlage, ggf. aufgeteilt auf verschiedene Stränge und Temperaturniveaus.

Heizkreise sind Gruppen zusammenhängender Bauteile eines Heizungssystems, welche sich über einen gemeinsamen Verteilerabgang definieren. Heizkreise bestehen i.d.R. aus mehreren meist parallel geschalteten Verbrauchern, sowie den zugehörigen Vor- und Rücklaufleitungen. Enthaltene Bauteile sind Rohrstücke, Bögen, Übergangs- und T-Stücke sowie Heizkörper. Heizkreise enthalten i.d.R. Regelungstechnik am Verteilerabgang und die Raumregelungseinrichtungen wie z.B. die Heizkörperthermostate. Eine detaillierte Darstellung von Heizkreisen erfolgt i.d.R. im Strangschemata einer Anlage (eine beispielhaften Ausschnitt zeigt Abb. 5). Im Gewerk Kälte existiert das analoge Konzept des Kühlkreises, welcher im Folgenden mitgemeint ist, auch wenn er nicht explizit genannt wird.

Heiz- und Kühlkreise

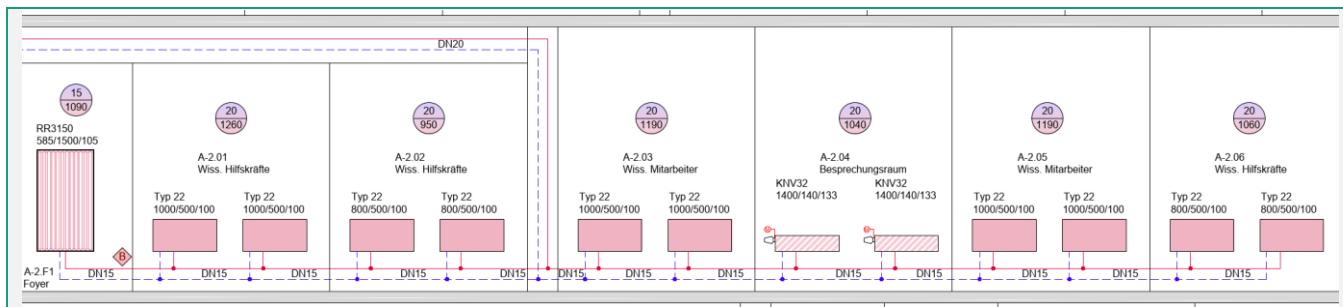


Abb. 5 beispielhafter Ausschnitt aus einem Strangschemata

Der obere Teil von Abb. 6 zeigt, wie aus dem Konstruktionsmodell weitere relevante Austauschdokumente und interne Dokumente abgeleitet werden. Diese spielen für die weiteren Betrachtungen in dieser Arbeit keine Rolle, da sie nicht direkt mit den Funktionsmodellen interagieren.

1 Motivation, Stand von Wissenschaft und Technik

1.1 Motivation

1.1.2 Planungsprozess der technischen Gebäudeausrüstung

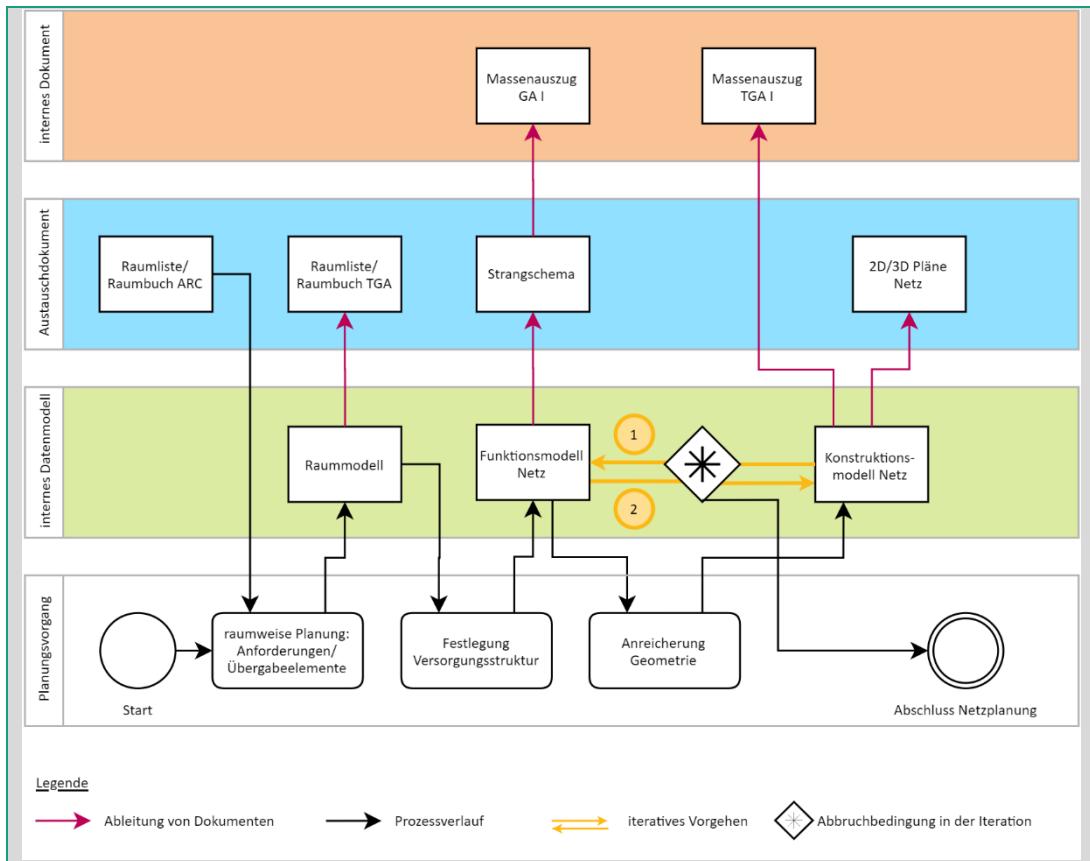


Abb. 6 Planungsprozess Netzplanung: Modelle und Dokumente im Planungsprozess

Die Zahlen werden in Abschnitt 1.5.1 thematisiert.

Die raumseitigen Anforderungen werden im Zuge der **Zentralplanung** um anderweitige Anforderungen ergänzt und es entsteht ein Anlagenkonzept wie in Abb. 8 dargestellt. Das Konzept ist im sog. Funktionsmodell enthalten. Dieses dient als Grundlage für Berechnung und Auslegung, danach erfolgt eine erste Konstruktion. Auch hier wird ggf. wieder ein iteratives Verfahren notwendig, um konstruktionsbedingte Änderungen am Funktionsmodell zurückzuführen. Aus der finalen Version des Funktionsmodells können die sog. „HLK³-Schalschemata“ (i.d.R. getrennt für Heizung und Kälte) abgeleitet werden.

Die raumseitigen Anforderungen werden dabei kummulierte als Heizkreise in die Zentralplanung übernommen, dementsprechend erfolgt auch die Darstellung im Schalschema mit Platzhaltern wie in Abb. 7 beispielhaft gezeigt. Der Heizkreisplatzhalter bietet somit die Schnittstelle zwischen Netz- und Zentralplanung, sowie auch zwischen den Funktionsmodellen von Netz und Zentrale.

³ Heizung, Lüftung, Kälte

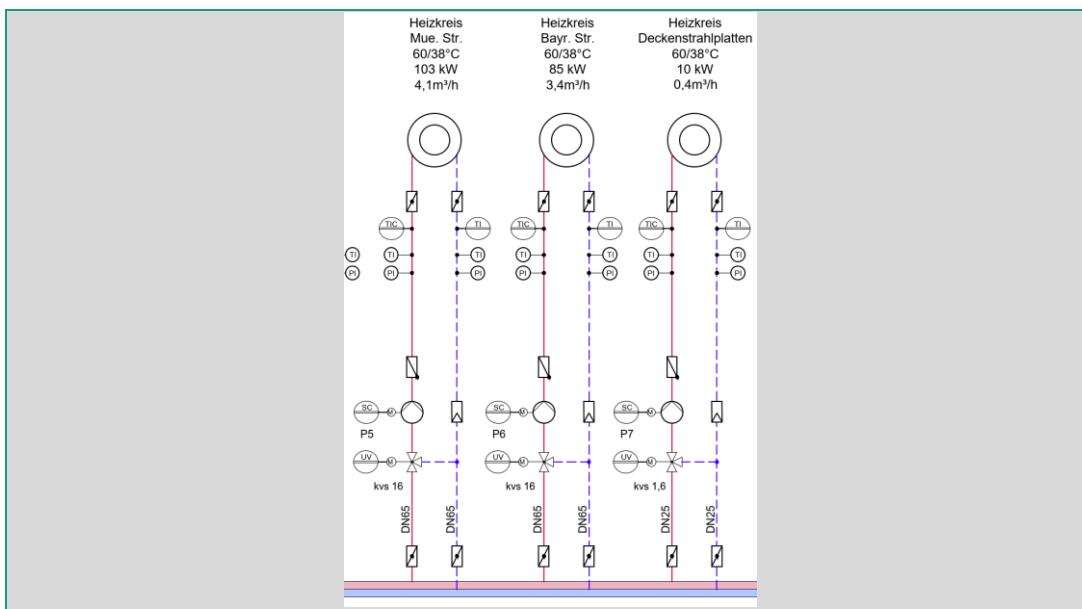


Abb. 7 beispielhafte Darstellung von Heizkreisen im Schaltschema

Viele Informationen in den beschriebenen **Austauschdokumenten** sind redundant. Es ist daher erstrebenswert, diese nur einmalig in einem „**Modell**“ zu halten und die Dokumente automatisch daraus abzuleiten. Dies gewährleistet die Konsistenz aller Dokumente, die bei manueller Erstellung nicht immer gewährleistet werden kann.

Wichtige **Dokumentarten** sind „Pläne“ und „Schemata“. Dabei enthalten diese Darstellungsformen eines Modells nur eine Auswahl der im zugrunde liegenden Modell enthaltenen Informationen. Strichstärken, -arten, -farben und Beschriftungen sind übliche Optionen um eine hohe Informationsdichte in derartigen Dokumenten zu erreichen. Es werden im Kontext der Planung zwei Arten von Modellen unterschieden: Konstruktions- und Funktionsmodelle.

Schemata werden entsprechend der oben beschriebenen unterschiedlichen Charakteristik unterteilt in **Schaltschema** (enthält als Ergebnis der „Zentralplanung“ die Erzeugung und Verbraucheranlagen) und **Strangschemata** (enthält die Netzverbraucher).

1 Motivation, Stand von Wissenschaft und Technik

1.1 Motivation

1.1.2 Planungsprozess der technischen Gebäudeausrüstung

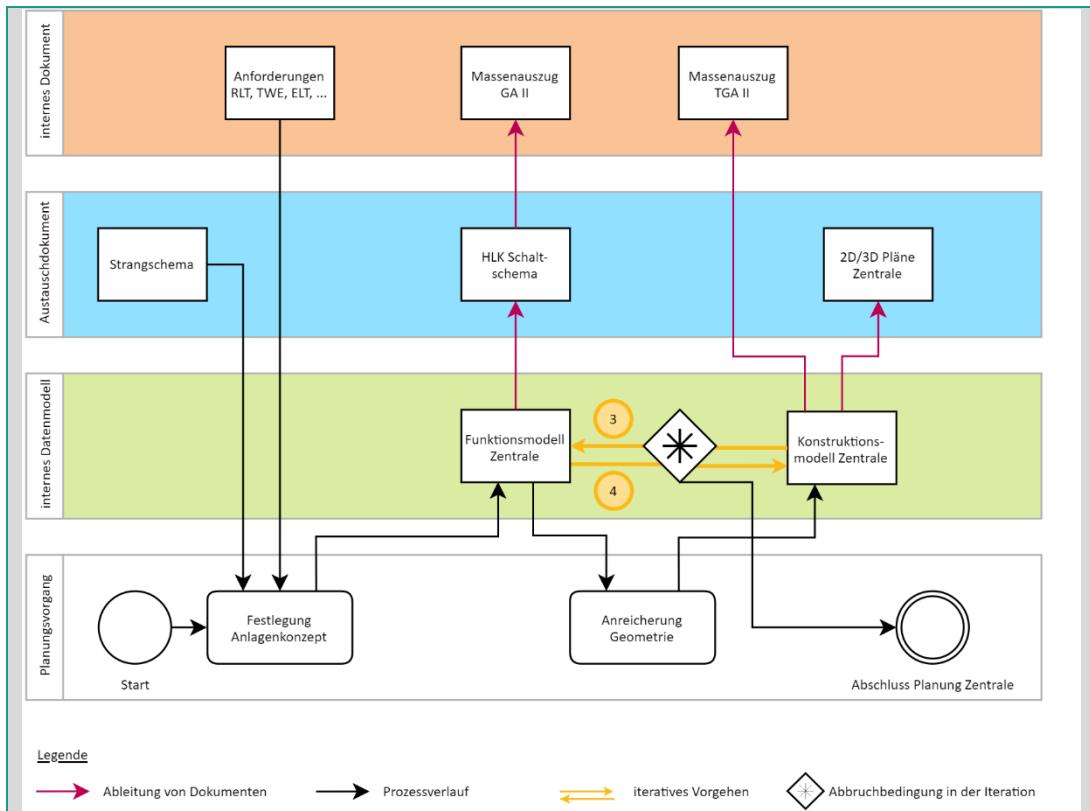


Abb. 8 Planungsprozess Zentralplanung
Die Zahlen werden in Abschnitt 1.5.1 thematisiert.

Modelle sind Abbildungen der Wirklichkeit zu einem bestimmten Zweck. Das **Konstruktionsmodell** enthält die geometrischen Informationen eines Systems für die Vorbereitung der Montage. Die geeignete Darstellungsform dazu ist der „Plan“, z.B. Grundrisse, Ansichten und Schnitte.

Das **Funktionsmodell** enthält demgegenüber alle funktionsrelevanten Informationen zu einem System für Untersuchungen zur Funktion und die Dimensionierung. Dazu gehören Angaben über die enthaltenen Bauteile eines Systems, sowie deren Eigenschaften in Hinblick auf die Funktion im Regelbetrieb, aber auch in Sonderbetriebsfällen wie Störung und Wartung. Schemata spielen als Darstellungsform eine wesentliche Rolle im Planungsprozess, sie enthalten i.d.R. alle wesentlichen Informationen zu einem System, außer der Geometrie – sie sind dementsprechend die zugehörige Darstellungsform des Funktionsmodells. Das Funktionsmodell kann als semantisch angereicherte Variante der Darstellungsform Schema begriffen werden.

Bauteile eines Systems können unterteilt werden in **schema-relevante** und schema-irrelevante Bauteile. Schema-relevant sind dabei vor allem Bauteile, welche Änderungen am Fluidstrom hervorrufen, die über einen konstanten Druckverlust⁴ hinausgehen z.B.

- Erzeuger (Zuführung von Leistung und daraus resultierende Temperaturänderung)
- Verbraucher (Leistungsentnahme und daraus resultierende Temperaturänderung)
- T-Stücke (Stromverteilung und -vereinigung)
- Regelarmaturen (variabler Druckverlustbeiwert).

⁴ Gemeint sind eigentlich Bauteile mit einem konstanten Druckverlustbeiwert ζ .

Die vorgenannten Bauteile werden im Folgenden als **strukturrelevant** bezeichnet. Schemarelevant sind darüber hinaus i.d.R. sicherheitsrelevante Bauteile, sowie Entleer-, Befüll- und Absperreinrichtungen. Funktionsmodelle dienen der Untersuchung der Funktion eines Systems. Dies erfolgt i.d.R. durch Berechnungen. Funktionsmodelle können demnach als **Berechnungsmodelle** ausgeprägt sein. Berechnungsmodelle, welche für die Verwendung in einem Simulationsprogramm inhaltlich und technisch vorbereitet sind, werden im Folgenden als **Simulationsmodelle** bezeichnet.

Jedes Planungsbüro setzt für die Erstellung von Konstruktions- und Funktionsmodellen auf eigene Werkzeuge und die dazugehörigen i.d.R. proprietären Dateiformate. Je nach verwendeter Software können beide Modelle auch in ein und derselben Datei realisiert sein. Funktionsmodelle liegen insb. für die Zentrale manchmal nur implizit im Kopf des Planers vor – in diesen Fällen wird das Schaltschema gezeichnet und nicht aus dem Funktionsmodell abgeleitet.

Die auf die Vorplanung folgenden Leistungsphasen Entwurfs- und Ausführungsplanung laufen i.d.R. ähnlich zur Vorplanung ab, jedoch mit höherem Detaillierungsgrad. Dargestellt sind in den Abb. 4 bis Abb. 8 nur interne Iterationen des Fachplaners für TGA. Externe Iterationen mit den anderen vorgenannten Beteiligten treten im Planungsprozess zwar häufig auf, ändern jedoch nichts an der grundsätzlichen Vorgehensweise. Der beschriebene Prozess muss dadurch lediglich mehrfach ausgeführt werden.

1.1.3 Rolle von BIM

Das Akronym BIM wird unterschiedlich verwendet: Während ihm vielfach die Bedeutung „Building Information Modeling“ zugeschrieben wird, kommt in dieser Arbeit die weitergehende Definition als „Building Information Management“ zur Anwendung, wie sie inhaltlich auch von (Scherer und Schapke 2014) und (Borrmann u. a. 2014) vertreten wird. „Modeling“ wird dabei in einer sehr weiten Definition als „Modellierung des gesamten mehrdimensionalen Informationsraumes“ (Scherer und Schapke 2014) verstanden. In diesem Sinne wurde auch in den Abb. 4ff bereits der Begriff **Datenmodell** für die informationsverwaltenden Entitäten genutzt.⁵

Hinsichtlich der technischen Realisierung des Informationsaustauschs wird unterschieden zwischen „open“ und „closed“ BIM. „Closed“ bezeichnet dabei Datenmanagementprozesse, welche innerhalb des Ökosystems eines Herstellers abgewickelt werden (z.B. Autodesk), „open“ BIM setzt hingegen auf die Verwendung offener Standards. Hinsichtlich der Anzahl der Beteiligten Akteure wird weiterhin zwischen „little“ (Anwendung nur innerhalb eines projektbeteiligten Büros) und „big“ (Anwendung über Bürogrenzen hinaus) BIM unterschieden.

Werden Modelle zwischen Planungsbeteiligten ausgetauscht (direkt oder über eine Plattform, wie z.B. in Abb. 10 dargestellt), so kann damit Doppelarbeit vermieden werden, denn oftmals sind die selben Informationen Grundlage verschiedener Planungsprozesse verschiedener Beteiligter. Es handelt sich hierbei also um einen ähnlichen Anwendungsfall, wie den oben beschriebenen Fall der Ableitung von Austauschdokumenten aus den Funktions- und Konstruktionsmodellen. Dazu werden Informationen zu einem bestimmten „Stand“ von anderen Beteiligten übernommen – oftmals ist der Architekt als Koordinierender des Planungsteams die Datenquelle. Die Mehrwerte durch vermiedene Doppelarbeit beim Informationsaustausch ändern jedoch nichts daran, dass jeder Prozessbeteiligte eine unterschiedliche Sicht auf dieselben Daten hat und daraus ggf. unterschiedliche Modellierungsanforderungen resultieren. Die inhaltlichen Verantwortlichkeiten

⁵ Modell = Abbildung der Wirklichkeit zu einem bestimmten Zweck

1 Motivation, Stand von Wissenschaft und Technik

1.1 Motivation

1.1.4 Simulieren im Planungsprozess

in der Planung ändern sich durch die effizientere technische Umsetzung mit der BIM-Methodik nicht.

BIM findet derzeit bereits verbreitet Anwendung in der Planung, laut (BauInfoConsult 2022) werden 42% aller Bauprojekte in Deutschland zumindest teilweise mit einem „digitalen Zwilling“ begleitet. Die Arbeitsweise ist dabei häufig noch dateibasiert. Die Dateien werden per Mail oder Common Data Environment (CDE) ausgetauscht. Die datenbankbasierte Zusammenarbeit auf Collaboration Platform ist derzeit nur als closed BIM in breiter Anwendung.

Patrick MacLeamy beschrieb 2004 den in Abb. 9 dargestellten Effekt einer Vorverlagerung von Planungsaufwänden in frühe Planungsphasen durch die BIM-Methode (vgl. blaue zu orangene Kurve). Für die Qualität der Planung ist dies günstig, da Änderungsbedarf in Phasen erkannt wird, in denen Änderungen noch ohne große Folgekosten möglich sind (rote Kurve befindet sich unter der grünen Kurve).

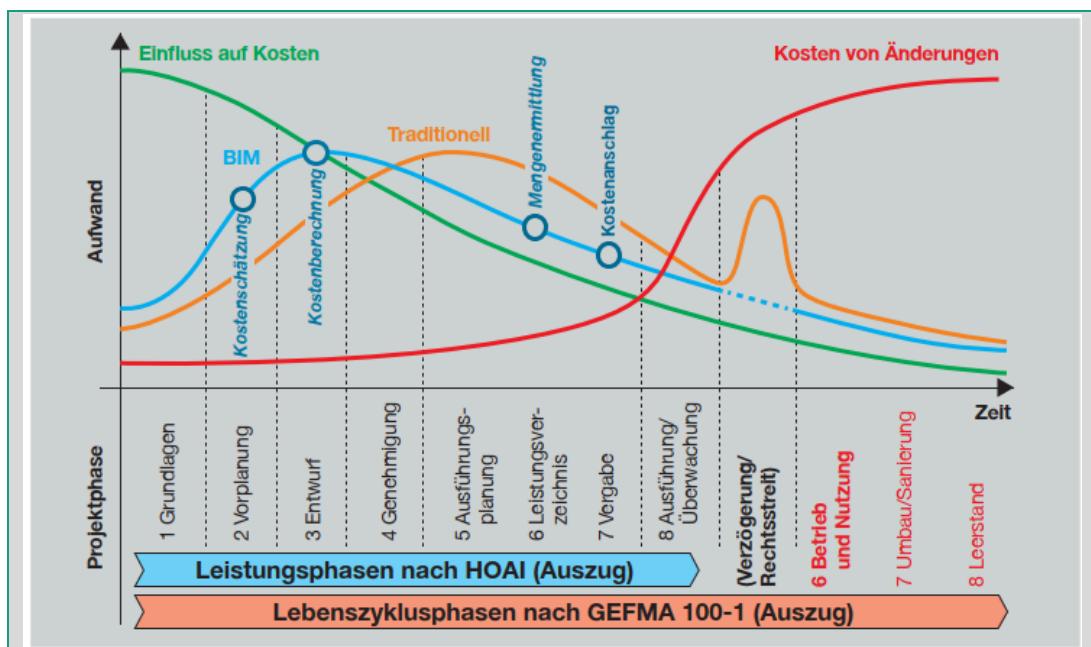


Abb. 9 McLeamy-Kurve

Bildquelle (van Treeck u. a. 2016)

1.1.4 Simulieren im Planungsprozess

1.1.1.1 Anwendungsfälle

Im Sinne der Verantwortlichkeit des Planers für ein funktionsfähiges Gebäude kann es notwendig sein, über die in nach HOAI und VDI 6026-1 geschuldeten Berechnungen hinaus in verschiedenen Planungsphasen weitere Untersuchungen durchzuführen. Denkbar sind beispielsweise:

- die Bewertung der Behaglichkeit in Räumen bei begrenzten Leistungen, Temperaturen, Volumenströmen für Räume und Raumlufttechnik,
- die Simulation von Fehlersituationen (z.B. Ausfall eines Erzeugers),
- die Computational Fluid Dynamics (CFD) Simulation für die Raumübergabe,
- Detailuntersuchungen zur Betonkernaktivierung etc.,
- die Betrachtung der Hydraulik bei Vermischung,
- eine Bewertung von Abweichungen zwischen idealer Regelung und realisierbarer Regelung (z.B. aufgrund nicht verfügbarer oder nicht ideal platzierbarer Sensoren, Totzeiten in der Hydraulik),

- Entwurf und Bewertung von Regelalgorithmen, z.B. für Heizung, Kühlung Sonnenschutz, Beleuchtung und Nachtlüftung.
- raumakustische Simulationen
- Brand- und Entrauchungssimulation
- Entfluchtungssimulation
- Energiebedarfsberechnung für Wirtschaftlichkeitsbetrachtungen und/oder energetische Nachweise (Energieausweis, Fördermittel)
- Anlagenauslegung
- Bewertung instationärer Betriebszustände von Erzeugern

Für viele dieser Untersuchungen sind Simulationsverfahren notwendig. **Simulationsverfahren** werden in dieser Arbeit in Abgrenzung zu klassischen **stationären Berechnungsverfahren** wie z.B. der Heizlastermittlung nach DIN EN 12831-1 verwendet. Sie zeichnen sich dadurch aus, dass mehrere Zeitschritte berechnet werden, die nicht unabhängig voneinander betrachtet werden können⁶. Oftmals handelt es sich um Jahressimulationen, jedoch ist auch die Küllastberechnung nach VDI 2078 mit ihrem 10-tägigen „Referenzzeitraum“ ein Simulationsverfahren. Am grundsätzlichen Ablauf der Planung, wie in den Abb. 4ff dargestellt, ändert sich durch den Einsatz von Simulationsverfahren nichts. Die Funktionsmodelle, welche Grundlage der Berechnungen sind (Simulationsmodelle), sind dann allerdings komplexer als Berechnungsmodelle für stationäre Berechnungsverfahren.

1.1.4.1 Mehrwerte

Mehrwerte entstehen bei Simulationsbetrachtungen (im Vergleich zu stationären Berechnungen) durch:

- A. Genauere Berechnung: Die statischen Berechnungsverfahren enthalten starke Vereinfachungen, die zu einer nennenswerten Überdimensionierung von Anlagen führen können. Damit einher gehen erhöhte Investitions- und Betriebskosten. Dies trifft beispielweise für die Heizlastberechnung zur Auslegung der zentralen Erzeuger und die Auslegung von Lüftungsregistern zu.⁷ Die Bewertung des Einflusses von schwankenden äußeren Faktoren wie Wetterbedingungen und intern aufgeprägten wechselnden Lasten ist nur mit Hilfe dynamischer Simulationsverfahren möglich.
- B. Bewertung zeitabhängiger Effekte: Die statischen Berechnungsverfahren enthalten so starke Vereinfachungen, dass die Bewertung bestimmter Effekte gar nicht erfolgen kann, beispielsweise die Nachtabsenkung von Sollraum- und/oder Heizmedienvorlauftemperaturen, effizientere Beleuchtung und Ausstattungsgegenstände oder der Einsatz geregelter Pumpen. Aufgrund der stationären Berechnungsansätze können insb. Speicherfüllstände nicht hinreichend berücksichtigt werden.
- C. Berechenbarkeit von Nicht-Standard-Konfigurationen:
 - Multivalente Systeme: Mit der Einbindung von erneuerbaren Energien nimmt die Verbreitung multivalenter Systeme immer mehr zu - bivale Erzeugungsanlagen mit einem dedizierten Grund- und Spitzenlastgerät reichen oft nicht mehr aus.
 - Besondere Betriebsweise: Während für bivale Anlagen die Vordefinition standardisierter Zusammenwirkungskonzepte (bivalent parallel / bivalent alternativ) noch möglich war, ist dies für multivalente Anlagen aufgrund des großen

⁶ Im Gegensatz dazu wird bei stationären Berechnungsverfahren immer nur ein Zeitpunkt betrachtet. Um die Funktionsfähigkeit der Anlagen dennoch jederzeit sicher zu stellen, muss es sich dabei um einen Worst-Case-Zeitpunkt handeln. Bereits die Ermittlung des Worst-Case-Szenarios ist jedoch ein fehleranfälliger Vorgang.

⁷ Die Küllastberechnung erfolgte nach VDI 2078 bis zum Jahr 2015 sehr ähnlich. Aufgrund der genannten Nachteile wurde hier bereits ein Simulationsverfahren mit der VDI 2078:2015 zum Stand der Technik erklärt.

Möglichkeitsraums nicht mehr realisierbar. Dementsprechend können auch keine Kennwerte dafür normiert werden.

- Geometrien: Für architektonisch innovative Gebäude sind die betrachteten Geometrien oft sehr unregelmäßig und die den genormten Verfahren zugrunde liegenden Vereinfachungen verlieren ihre Gültigkeit.
- D. Effiziente Regelungskonzepte: Ein Großteil der Effizienz in Anlagen entsteht durch eine anforderungsgerechte Regelung im Betrieb: Regelalgorithmen können für multivalente Systeme nur mit Simulationsverfahren sinnvoll entworfen und bewertet werden.
- E. Zusätzliche Mehrwerte entstehen, wenn ein und dasselbe Modell mit lediglich anderer Parametrierung für verschiedene Fragestellungen, z.B. die Berechnung des Sommerlichen Wärmeschutz und die Kühlastermittlung (zutreffend für Gebäudesimulations-Modelle), genutzt werden. Eine Nachnutzung der Modelle in der Betriebsphase (siehe Abschnitte 6.2 und 6.3) sichert zusätzliche Mehrwerte.

1.1.4.2 Hemmnisse

Jedoch gibt es auch Hemmnisse beim Einsatz von Simulationsverfahren. Dazu gehören:

- F. Aufwand für Modellerstellung: Es entsteht erheblicher Aufwand für die Modellerstellung. Jedoch müssten viele Randbedingungen auch für die stationären Berechnungsverfahren ermittelt werden.
- G. Nachvollziehbarkeit: Simulationsverfahren sind schwerer nachzuvollziehen als stationäre Berechnungen. Es ist daher i.d.R. Expertenwissen notwendig, um sie anzuwenden.
- H. Normkonformität: Simulationsverfahren sind oft nicht normkonform. Dadurch entsteht eine formelle und ggf. juristische Hürde. Im Sinne einer „allgemein anerkannten Regel der Technik“ (a.a.R.d.T.) ist der Einsatz von Simulationsverfahren dadurch jedoch nicht ausgeschlossen, sondern – im Gegenteil – unter Umständen (siehe Anwendungsfälle in Abschnitt 1.1.1.1) sogar angezeigt. Die HOAI trägt dem Rechnung, indem Simulationsverfahren als „Besondere Leistung“ enthalten sind.
- I. Abstimmungsaufwand: Aufgrund der Komplexität des Verfahrens sind zahlreiche Randparameter notwendig, welche ggf. mit der Bauherrschaft abzustimmen sind. Auch aufgrund der o.g. formellen Hürde besteht erhöhter Beratungsbedarf.
- J. Honorierung: Simulationsverfahren sind keine Grundleistung nach HOAI. Entsprechend den dort festgelegten Honorargrundsätzen sind Verfahren, die zu verminderten Anlagengrößen und damit Investitionssummen führen, sogar kontraproduktiv für den Planer, da kleiner ausgelegte Anlagen das Honorar schmälern.

Man erkennt, dass die letztgenannten Hemmnisse (G bis J) durch die seltene Nutzung der Verfahren entstehen. Ein verringelter Aufwand für die Modellerstellung würde die einzige objektive Hürde (F) beseitigen und damit dazu beitragen, dass die Mehrwerte von Simulationsuntersuchungen bei mehr Gebäuden zur Anwendung kommen könnten.

Die erhöhten Aufwände einer Simulation haben eine ähnliche Charakteristik, wie sie die McLeamy-Kurve (Abb. 9) für die BIM-Methodik beschreibt: Aufwände werden vorverlagert in eine Phase, in der Design-Änderungen noch günstiger möglich sind. Im Unterschied zur McLeamy-Kurve bei der BIM-Methodik, werden bei Simulationsbetrachtungen Aufwände aus der Inbetriebnahme- und Betriebsphase in die Planungsphase verlagert und somit in die Domäne ganz anderer Projektbeteiligter verschoben, was die Honorierungsproblematik noch verschärft. Aufgrund der Probleme bei der Honorierung sind Simulationsverfahren derzeit vorrangig bei Sonderbauten in der Anwendung, wo ohnehin extra honorierte Ingenieurleistungen vorab vereinbart werden.

1.2 Status Quo BIM

Die Schlüsselrolle der Funktionsmodelle für den Planungsprozess wurde bereits in Abschnitt 1.1.2 thematisiert. Sie können dazu beitragen, den Aufwand für die Erstellung von Simulationsmodellen signifikant zu verringern (Punkt F in obiger Auflistung) bzw. können Simulationsmodelle die Rolle des Funktionsmodells übernehmen und damit die Nachnutzung der Modelle (Punkt E in obiger Auflistung) gewährleisten.

Im Planungsprozess werden Informationen i.d.R. von anderen Beteiligten zu einem bestimmten Zeitpunkt mit einem Arbeitsstand (im Folgenden: *Stand*) übernommen, anschließend werden eigene Informationen angereichert und erzeugt. Eine Teilmenge davon wird wiederum von Anderen zu einem neuen Stand übernommen. Die Austauschprozesse von Ständen sollen durch BIM vereinfacht werden – Doppelarbeit möglichst vollständig vermieden werden. Dazu wird an den Schnittstellen/beim Austausch auf Standards gesetzt. BIM regelt ausschließlich die Arbeitsweise an den Datenübergaben, nicht dazwischen. Es steht den Beteiligten frei, Fachsoftware mit proprietären Datenformaten zu nutzen, solange am Ende die vereinbarten Schnittstellen bedient werden können. Die grundsätzlichen Konzepte dazu sind in DIN EN ISO 19650-1 dargestellt.

Notwendig für den Datenaustausch sind einheitliche Datenschemata (z.B. IFC4) und Dateiformate (z.B. IFC STEP). Da das IFC-STEP-Format das Konzept der „Referenzierung“ fremder Informationen⁸ zwar ansatzweise vorsieht, dies aber praktisch nicht verwendet wird, enthalten die Austauschdateien oftmals eine Mischung eigener (selbst erzeugter) und fremder (vormals übernommener) Informationen. Möglichkeiten, die inhaltlichen Verantwortlichkeiten auch technisch abzubilden, sind vorhanden und werden in Abschnitt 2.1.8.2 beschrieben. Sie haben bisher jedoch keine breite Anwendung in der Praxis gefunden. Der in DIN EN ISO 19650-1 definierte Informationsmanagementprozess, welcher parallel zum Lebenszyklus eines Bauwerks besteht, behandelt diese Thematik.

1.2.1 Normung

In den letzten 25 Jahren sind zahlreiche Standards im Bereich BIM entstanden. Wesentliche nationale Normungsorgane sind das Deutsche Institut für Normung (DIN) mit seinen internationalen Pendant CEN und ISO. Im Bereich der Vornormung ist der Verein buildingSMART, in Deutschland oft in Kooperation mit dem Verein deutscher Ingenieure (VDI) tätig.

Nach (van Treeck u. a. 2016) können die Normen im Bereich BIM in vier Gruppen eingeteilt werden:

- Datenstandards: definieren, wo Informationen abgelegt bzw. wie diese technisch ausgetauscht werden, z.B. über IFC-Dateien
- Modellstandards: definieren, welche Informationen im Planungsprozess ausgetauscht werden
- Methodenstandards: definieren, wie Informationen erarbeitet, geprüft und ausgewertet werden
- Managementstandards: definieren Verantwortlichkeiten, Zeitpunkte und Qualitätsanforderungen

1.2.1.1 Datenstandards

Tab. 1 listet Normen im Bereich der Datenstandards für den Austausch von Produkt- und Herstellerdaten auf. Wesentlich sind dabei die Industrial Foundation Classes (IFC), welche sich,

⁸ analog zu XREFs in der AutoCAD-Welt

1 Motivation, Stand von Wissenschaft und Technik

1.2 Status Quo BIM

1.2.1 Normung

zusammen mit dem zugehörigen IFC-STEP-Format, als führendes OpenBIM-Datenschema⁹ etabliert haben. In Abschnitt 2.1 wird IFC vertieft beschrieben. Weiterhin existieren Standards für den Austausch mehrerer zusammengehöriger Dokumente, ggf. incl. Verlinkung untereinander: Dafür wurden der ICDD (Information Container for Data Delivery) in DIN EN ISO 21597 und der MMC (MultiModellContainer) in DIN 18290-1 definiert. Für den Austausch von Produktdaten (Katalogdaten) existiert die langjährig etablierte VDI 3805, welche zukünftig von der, erst in Teilen existierenden, internationalen DIN EN ISO 16757 abgelöst werden soll. Dieser Standard befasst sich sowohl mit Geometrie, als auch mit alphanumerischen Informationen und Kennlinien/Funktionen.

Tab. 1 Normen BIM Datenstandards

Standard	Bezeichnung	Relevante Versionen	Inhalt
DIN EN ISO 16739-1	Industry Foundation Classes (IFC) für den Datenaustausch in der Bauwirtschaft und im Anlagenmanagement	2018	IFC4.0.2.1
		2024	IFC4.3.2
DIN EN prEN 17549-1,2	Building Information Modeling – Datenstruktur nach EN ISO 16739-1 für den Austausch von Datenvorlagen und Datenblättern für Bauobjekte	2022-07, 2023-09 2022-XX	IFC4.0.2.1
DIN 18290	Verlinkter BIM-Datenaustausch von Bauwerksinformationsmodellen mit weiteren Fachmodellen Teil 1: Verlinkter Datenaustausch mehrerer Fachmodelle beim Building Information Modeling (Multimodell-Container)	2023-11	Multimodellcontainer MMC basierend auf XML, alternative Implementierung zu DIN EN ISO 21597-1
	Teil 2: BIM-LV-Container Teil 3: BIM-Kosten-Container Teil 4: BIM-Abrechnungs-Container	2023-11	BIM-LV-Container BIM-Kosten-Container BIM-Abrechnungs-Container
			Zitiert von VDI 2552 Blatt 3:2018-05 „Mengenermittlung BIM“ und GEFMA 470:2017-09
DIN EN ISO 21597-1	Informationscontainer zur Datenübergabe - Austausch-Spezifikation - Teil 1: Container	2021-07	ICDD alternative zu DIN 18290-1
VDI 3805	Produktdatenaustausch in der technischen Gebäudeausrüstung		Wird blattweise ersetzt durch DIN EN ISO 16757
DIN EN ISO 16757	Datenstrukturen für elektronische Produktkataloge der Technischen Gebäudeausrüstung Teil 1: Konzepte, Achtitektur, Modelle	2019-10	Basiert auf „Terms“ gemäß ISO 12006-3 Statische und dynamische Merkmale (Kennlinien) können enthalten sein Kann auch Zubehör und Komponenten enthalten Webplatform zur Bereitsstellung existiert bereits: www.bim4hvac.com
	Teil 2: Geometrie	2019-10	Basiert im Wesentlichen auf den Geometriedarstellungen in DIN EN ISO 16739, ergänzt diese um Liniendarstellung Verschiedene Geometrien (z.B. Störkontur, Wartungsraum können dargestellt werden)
	Teile 3-5,010ff	Noch nicht erschienen	Ersetzen stückweise die Blätter der VDI 3805

1.2.1.2 Modellstandards

Mit den in Tab. 1 beschriebenen Standards sind die technischen Möglichkeiten beschrieben, Informationen zu übertragen, jedoch nicht die Frage geklärt, welche Information zu welchem Zeitpunkt notwendig ist. Dies wird mit Modellstandards (siehe Tab. 2) erreicht.

LOD („level of detail“ bzw. „level of development“), LOG („level of geometry“), LOI („level of information“) sind Begriffe, die den Grad der Modelldetaillierung beschreiben. Dabei wird der Detaillierungsgrad i.d.R. unterschieden zwischen geometrischen und nicht geometrischen Daten, da beides u.U. stark differieren kann. Die Begriffe haben den Anspruch, ähnlich den früher üblichen Definitionen von Maßstäben wie z.B. 1:100, 1:50, 1:10 wenige vordefinierte Detaillierungsgrade vorzusehen und diese möglichst auch einer Planungsphase zuzuordnen. Sie sind derzeit in keiner

⁹ Zur Definition von Open und Closed BIM siehe S. 35

in Deutschland gültigen Norm definiert. Definitionen wurden z.B. vom US amerikanischen BIM-Forum („Level of development (LOD) specification“ 2013) und der Australischen NATSPEC („LOD Australian NATSPEC National BIM Guide“ 2011) vorgenommen. Beide enthalten fünf bzw. sechs vordefinierte Level LOD 100...500. Die Definition des BIMForum fokussiert stark auf das LOG, während die NATSPEC auch umfangreiche Anforderungen an die nicht-geometrischen Daten stellt.

Die DIN EN ISO 29481 Teil 1 sieht jedoch vor, dass die Modellanforderungen prozessbasiert und ggf. projektspezifisch festgelegt werden. Dies erscheint insb. in Hinblick auf die Datendetaillierung sinnvoll, da die notwendigen Informationen stark vom Anwendungsfall abhängen. Derartige Anwendungsfälle sind z.B. in den Unterblättern der VDI 2552 Blatt 11 definiert.

Die anwendungfallspezifischen Austauschanforderungen werden oft in der Auftraggeber-informationsanforderung (AIA) zusammengefasst und als Exchange Information Requirements (EIR) bezeichnet.

1 Motivation, Stand von Wissenschaft und Technik

1.2 Status Quo BIM

1.2.1 Normung

Tab. 2 BIM Modell-, Methoden- und Managementstandards

Standard	Bezeichnung	Relevante Versionen	Inhalt
DIN EN ISO 29481	Bauwerksinformationsmodelle - Handbuch der Informationslieferungen Teil 1: Methodik und Format	2018-01	IDM MVD
	Teil 2: Interaktionsframework	2017-09	Beschreibt die Geschäfts- und Kommunikationsprozess im Bau
	Teil 3: Datenschema und Klassifikation	Entwurf 2021-10	Maschinenlesbares Format für IDMs
VDI 2552	Blätter 1-7		BIM: Grundlagen, Begriffe, Mengenermittlung, Datenaustausch, Datenmanagement, Betrieb, Prozesse
	Blatt 11, Unterblätter 1-9		Informationsaustauschanforderungen
DIN EN ISO 17412	Bauwerksinformationsmodellierung – Informationsbedarfstiefe Teil 1: Konzepte und Grundsätze	2021-06	Verfahren zur Beschreibung der Informationen in den AIA
ISO 12911	Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — Framework for specification of BIM implementation	2023-02	Hinweise zur Erstellung von BAP und IDM, so dass sie „testbar“ sind
DIN EN ISO 12006	Teil 2: Struktur für die Klassifizierung	2020-07	Hochbau - Organisation des Austausches von Informationen über die Durchführung von Hoch- und Tiefbauten
	Teil 3: Struktur für den objektorientierten Informationsaustausch	2022-10	Definiert wie ein „Wörterbuch“ zu definieren ist darauf basiert z.B. BSDD EXPRESS und UML, derzeit noch kein TTL/OWL
DIN EN ISO 23387	Bauwerksinformationsmodellierung (BIM) - Datenvorlagen für Bauobjekte während des Lebenszyklus eines baulichen Vermögensgegenstandes - Konzepte und Grundsätze	2020-12	Produktdatenvorlagen Regeln für die Verknüpfung von Datenvorlagen mit IFC und Klassifikationssystemen nach 12006-3
DIN EN ISO 23386	Bauwerksinformationsmodellierung und andere digitale Prozesse im Bauwesen – Methodik zur Beschreibung, Erstellung und Pflege von Merkmalen in miteinander verbundenen Datenkatalogen	2020-11	Regelt die Definition von im Bauwesen verwendeten Merkmalen und eine Methodik für die Erstellung und Pflege solcher Merkmale Legt Metadaten für Merkmale fest
DIN EN ISO 19650	Organisation und Digitalisierung von Informationen zu Bauwerken und Ingenieurleistungen, einschließlich Bauwerksinformationsmodellierung (BIM) - Informationsmanagement mit BIM Teil 1: Begriffe und Grundsätze	2019-08	Grundlagen zu BIM z.B. Begriff EIR
	Teil 2: Planungs-, Bau- und Inbetriebnahmephase	2019-08	Beschreibt allgemein acht verschiedene Informationsmanagementprozesse im Lebenszyklus
ISO 22263	Organization of information about construction works — Framework for management of project information	2008-01	Definition von Begrifflichkeiten in den Lebenszyklusphasen eines Bauwerks
DIN SPEC 91391	Gemeinsame Datenumgebungen (CDE) für BIM-Projekte - Funktionen und offener Datenaustausch zwischen Plattformen unterschiedlicher Hersteller - Teil 1: Module und Funktionen einer Gemeinsamen Datenumgebung	2019-04	Vergleichbarkeit von CDE aufgrund der Definition funktionaler Anforderungen (obligatorische und optionale)

In DIN EN ISO 12006 wird eine Vorgehensweise für die Erstellung von Klassifikationen und Terminologien definiert. Diese bilden die Basis für die Erstellung von Katalogen gemäß DIN EN ISO 23386 und 23387.

1.2.1.3 Methoden- und Managementstandards

Die DIN EN ISO 19650-1 erläutert die grundsätzlichen Zusammenhänge und Ideen des Informationsmanagements bei Bauwerken mit der BIM-Methode. Parallel zum Lebenszyklus des Bauwerks wird dabei ein Informationsmanagementprozess definiert. Zu definierten Zeitpunkten entstehen dabei zwischen ausgewählten Projektbeteiligten Austauschinformationsanforderungen (Exchange Information Requirements (EIR)). Diese werden durch die Informationsbedarfstiefe nach DIN EN ISO 17412 und das Information Delivery Manual gemäß DIN EN ISO 29481 näher beschrieben. ISO 22263 liefert dafür Begriffsdefinitionen. ISO 12911 gibt Hinweise zur Erstellung

von BIM-Abwicklungsplänen (BAP) und nimmt dabei Bezug auf DIN EN ISO 29481, sowie DIN EN ISO 19650-1.

Die Richtlinienreihe VDI 2552 ergänzt die oben beschriebenen Normen anwendungsnahe mit einem speziellen Fokus auf den deutschen Markt. Neben allgemeinen Grundlagen und Begriffen (Blätter 1 bis 7), werden Qualifikationen definiert (Blätter 8.1 und 8.2) sowie in Blattreihe 11 konkrete Informationsaustauschanforderungen definiert. Dabei werden verschiedene Anwendungsfälle in den Blick genommen, wie z.B. Schlitz- und Durchbruchsplanung in Unterblatt 11.2 und die Bauphysik in Unterblatt 11.6¹⁰.

1.2.2 Softwarewerkzeuge für die Bearbeitung von IFC-Modellen

1.2.2.1 Systematisierung der Werkzeuge

In diesem Abschnitt wird dargestellt, welche Softwarewerkzeuge für die Arbeit mit BIM-Modellen nach dem IFC-Standard im Kontext der Planung zum Zeitpunkt der Erstellung dieser Arbeit marktverfügbar sind. Es werden vier Arten von IFC-Werkzeugen unterschieden:

- **Autorentools:** dienen der (erstmaligen) Erstellung von Modellen und haben i.d.R. einen starken Fokus auf 3D-Geometrie, sie werden auch als CAD-Software bezeichnet. Große Verbreitung jenseits der Forschungscommunity haben hier lediglich die kommerziellen Werkzeuge der großen CAD-Anbieter. Die entstehenden Modelle liegen i.d.R. in einem proprietären Format vor, können aber als IFC exportiert werden. Dafür sind umfangreiche Einstellungen oftmals möglich und nötig.
- **IFC-Viewer:** zeichnen sich dadurch aus, dass ihr Fokus nicht auf der Erstellung, sondern auf der Prüfung, Sichtung und Manipulation von Modellen liegt. Sie haben demnach einen gegenüber den Autorentools reduzierten Leistungsumfang: Werkzeuge zur Manipulation der Geometrie fehlen in IFC-Viewern idR. Viele dieser Programme haben ein natives Format, können jedoch IFC sowohl importieren als auch exportieren.
- **IFC-Bibliotheken:** Diese Softwarebibliotheken ermöglichen die Interaktion mit IFC-Daten mit Hilfe von selbstgeschriebenem Quellcode. Von kleinen Abfragen bis zur Eigenentwicklung vollständiger Anwendungen ist damit alles möglich.
- **Kollaborationstools:** ermöglichen die in DIN EN ISO 19650-1 beschriebene modellbasierte Zusammenarbeit.

Im Sinne der Nachvollziehbarkeit und Freiheit der Forschung sollen in dieser Arbeit möglichst FLOSS¹¹-Tools oder wenigstens kostenlose Werkzeuge genutzt werden. In Tab. 3 werden ausgewählte Werkzeuge daher anhand dieses Kriteriums eingruppiert.

¹⁰ Dieses Blatt erscheint voraussichtlich im Frühjahr 2025 als Gründruck und entstand und federführender Beteiligung der Autorin dieser Arbeit.

¹¹ FLOSS = Free/Libre and Open Source Software

Tab. 3 Übersicht BIM-Tools

Art des Werkzeugs	kommerziell	Nicht kommerziell
Autorentool	Allplan Revit Archicad Sketchup	BlenderBIM FreeCAD
IFC-Viewer	Solibri simpleBIM BIMvision (Basisversion)	FZK KIT-Viewer IfcPlusPlus
IFC-Bibliotheken	Xeokit IFC Tools groovyIFC	Ifcopenshell Bimfit ifcPlusPlus Billie xBIM Toolkit NIST IFC File analyzer
Kollaborationstools	BIMdata.io	Bimserver IFCWebServer.org

Ein Vergleich der freien Autorentools BlenderBIM und FreeCAD für die halbautomatisierte Erstellung von Gebäudemodellen findet sich in (Challagulla 2023). Im Vergleich zu den kommerziellen Tools zeigte sich für beide Tools eine schlechtere Handhabbarkeit. Es soll daher möglichst vollständig auf die Verwendung von Autorentools (freie oder kommerzielle) verzichtet werden.

IFC-Viewer und Softwarebibliotheken sind von hoher Relevanz für die in dieser Arbeit beschriebenen Anwendungsfälle und werden daher im Folgenden vergleichend betrachtet, um eine Entscheidung für die in Kapitel 3.11 beschriebene Implementierung herbeizuführen. Kollaborationstools haben hingegen keine Relevanz und werden daher nicht vertieft behandelt.

1.2.2.2 IFC-Bibliotheken¹²

Für die Entwicklung eigener Softwarelösungen mit IFC-Unterstützung eignen sich die im Folgenden genannten Bibliotheken und Werkzeuge. Zunächst befinden sich darunter Libraries für verschiedene Programmiersprachen, welche die Basis für die Entwicklung eigener Softwareanwendungen sein können: BimFit, Billie, ifcPlusPlus, xBIM-Toolkit. Diese Bibliotheken unterscheiden sich hinsichtlich ihres Funktionsumfang insb. hinsichtlich den Möglichkeiten zum grafischen Rendering. ifcengine, ifcopenshell, Billie sind speziell für das grafische Rendering vorgesehen, enthalten jedoch auch Basisfunktionalitäten um Modelle alphanumerisch und semantisch auszulesen.

Weiterhin zählen dazu die Anwendungen IFCWebserver und BIMserver, welche entweder als Web- oder als Desktopanwendung für unterschiedliche Betriebssysteme vorliegen. Da diese OpenSource vorliegen und die Lizenz es zulässt, können Teile davon (auch in anderem als den ursprünglich vorgesehenen Anwendungen) zum Einsatz kommen.

Eine dritte Kategorie stellen die Software Development Kits (SDK) dar. Als solche werden definierte Schnittstellen zu kommerziellen Produkten bezeichnet: BIMdata.io, simpleBIM API – diese werden nicht weiter betrachtet, da bereits mit FLOSS-Werkzeugen ein ausreichender Funktionsumfang realisiert werden kann.

Tab. 4 stellt die verschiedenen BIM-Bibliotheken gegenüber. Ausgeschlossen für die weitere Verwendung wurden Projekte, die nicht mehr aktiv gepflegt werden oder über keine brauchbare Dokumentation verfügen.

¹² Einige Passagen in diesem Abschnitt wurden von der Autorin bereits veröffentlicht in (Eckstädt, Huang, u. a. 2023) Kap 1.5.3 und 1.5.4

Tab. 4 IFC-Bibliotheken

	Sprache(n)	Lizenz	Dokumentation (Schulnote)	Funktionsumfang	Website bzw. zugehörige wiss. VÖ	GitHub-Repository und Bewertung der Community-Aktivität
BIMFit	Java	AGPL	4	nur lesen/filtern	(CIB 2014)	Kein GitHub repository, inaktiv
IfcOpenShell	Python, C++	LGPL	2	lesen, schreiben, Geometrie rendern	(Krijnen 2011)	github.com/ifcopenshell/ifcopenshell , aktiv
IFCWebServer	Ruby	k.A.	5	nur lesen/filtern	(Ismail 2011)	github.com/ifcwebserver/ifcwebserver , nur 1 aktive Person
BIMServer.org	Java	AGPL	2	lesen, schreiben	(Beetz u. a. 2010)	github.com/opensourceBIM/BIMserver , aktiv, Einbindung bei buildingSMART
Billie	k.A.	k.A.	1	nur Geometrie	(Tauscher 2016)	Kein GitHub repository, inaktiv
ifcPlusPlus	C++	MIT	5	lesen	(Gerold 2014)	github.com/ifcquery/ifcplusplus , aktiv
xBIM toolkit	C#, C++, .net	CDDL	2	lesen, schreiben, Geometrie	(Lockley 2007)	github.com/xBimTeam/XbimEssentials , aktiv

Mit Blick auf den Funktionsumfang, die unterstützten Programmiersprachen und Unterstützung durch Dokumentation und Community wird für die weitere Arbeit ifcopenshell benutzt.

1.2.2.3 IFC-Viewer

IFC-Viewer visualisieren Geometrie und Daten eines Modell, sie dienen damit der Modellprüfung. Weiterhin können auch zusätzliche Informationen im IFC-Tool ergänzt werden. Jedoch sollten Daten, welche die Gebäudegeometrie betreffen, nur in sog. Autorentools (s.o.) vorgenommen werden. Andernfalls kann es zu Inkonsistenzen kommen, wenn z.B. eine Bemaßung vom BIM-Tool für ein Bauteil verändert wird, aber die angrenzenden Objekte nicht passend dazu verändert werden - bei Autorentools ist sichergestellt, dass das 3D-Modell konsistent ist. Tab. 5 gibt eine Übersicht über gegenwärtig marktverfügbare und verbreitete IFC-Viewer.

Tab. 5 Marktübersicht IFC-Viewer

Tool Bezeichnung und aktuelle Version (Stand März 2024)	Betriebs-system	Hersteller	Kosten	Website
simpleBIM 10	Win	Datacubist Oy (SWE)	kostenpflichtig	simplebim.com/
BIM Vision 2.27.7	Win	Datacomp (POL)	Basisversion kostenlos	bimvision.eu/de/
Solibri Office	Win	Solibri (FIN)/Nemetschek	Basisversion kostenlos	www.solibri.com/
BIMCollab ZOOM	Win, MacOS	KUBUS	kostenpflichtig	www.bimcollab.com/de/products/bim_collab-zoom/
FZK-Viewer 6.5.1 (eingestellt)	Win	KIT (GER)	Freeware	www.iai.kit.edu/1302.php
KIT-Viewer 7.1 (Nachfolger)	Win	KIT (GER)	Freeware	www.iai.kit.edu/1302.php

Tab. 6 enthält die Kriterien für die Auswahl eines IFC-Viewers. Darunter sind einige, die sich auf die Unterstützung spezieller, in dieser Arbeit relevanter, Aspekte des IFC-Standards beziehen. Die technischen Grundlagen dazu werden in Abschnitt 2.1 erläutert, die Anwendung wird in Kapitel 5 thematisiert.

Tab. 6 Vergleich von IFC-Viewern anhand ausgewählter Softwarefunktionen

Tool (zum Test genutzte Version)	Änderung Parameter	IfcVirtualElement	IfcZone	IfcGroup	IfcSystem anzeigen	2D Context	Ladedauer in s*	Bedarf Arbeitsspeicher in GB*
simpleBIM 9.1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> auch: erzeugen	<input type="checkbox"/>	85 s	0,9
BIM Vision	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	50 s	1,2
Solibri	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	140 s	6,2
FZK-Viewer 6.4.3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	> 10min	4,3
KIT-Viewer 7.1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	220 s	5,3

* Für das in Abb. 51 dargestellte Beispielmodell (Dateigröße des IFC-STEP-Files 69 MB)

Man erkennt, dass sich die Tools insb. hinsichtlich der Unterstützung eines zweidimensionalen Darstellungskontexts unterscheiden, welcher in Hinblick auf die zu untersuchenden Thesen der

Arbeit (siehe Abschnitt 1.5.2) von herausgehobener Bedeutung ist. Im weiteren Verlauf dieser Arbeit wird daher vornehmlich der KIT-Viewer verwendet.

1.2.3 Forschung

1.2.3.1 Multimodelle, (distributed) Digital Twin

Abb. 10 zeigt die Veränderungen des Kommunikationsablaufs in der Planung durch die Einführung von BIM. Zahlreiche bilaterale Absprachen zwischen den Planungsbeteiligten werden dabei durch die Arbeit an einem gemeinsamen Zentralmodell ersetzt. Die Darstellung unterschlägt dabei die unterschiedlichen fachlichen (und juristischen) Verantwortlichkeiten der beteiligten Akteure. Das „Zentralmodell“ kann daher kein monolithisches Artefakt wie z.B. eine Datei sein, sondern muss den Verantwortlichkeiten Rechnung tragen. Datenbankbasierte Zusammenarbeit auf Common Data Environments (CDE) oder verschiedene Fachmodelle tragen diesem Umstand in der aktuellen Planungspraxis Rechnung.

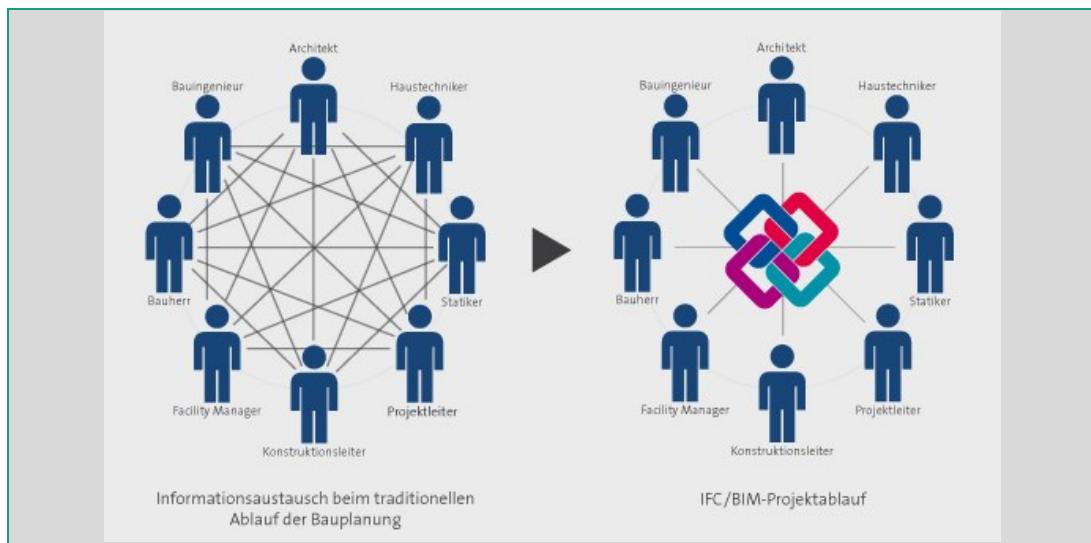


Abb. 10 Vergleich des Kommunikationsablaufes nach traditioneller Planung und Planung nach der BIM-Methode

Bildquelle: („Graphisoft Building Systems GmbH, „OPEN BIM und IFC,‘ Data Design System GmbH“, o. J.)

Scherer prägte 2014 für die Einheit aus zusammengehörigen Fachmodellen den Begriff „Multimodell“ (Scherer und Schapke 2014). Für den Austausch derartiger Modelle wurde inzwischen der MultiModellContainer als DIN 18290-1 normiert, eine alternative Implementierung ist der Information Container for Data Drop in DIN EN ISO 21597. Beides adressiert den Austausch von Multimodellen als große Artefakte zu definierten Zeitpunkten.

Das Konzept des Multimodells sieht es insb. auch vor, dass Bestandteile in unterschiedlichen Datenformaten und -schemata vorliegen können, neben IFC-Dateien können z.B. GAEB-Dateien (Leistungsverzeichnisse) und Modelica-Dateien (Simulationsmodelle) Bestandteile eines Multimodells sein. Die in Abschnitt 1.1.2 eingeführten Funktions- und Konstruktionsmodelle können ebenfalls als Bestandteile eines Multimodells aufgefasst werden.

Für den Begriff „Digital Twin“ existiert keine einheitliche Definition, im Allgemeinen wird darunter eine virtuelle Abbildung eines Objekts aus der realen Welt verstanden – er kann damit als Synonym für den Begriff „Modell“ aufgefasst werden. Als logische Weiterführung des (Multi)Modells bzw. „Digital Twin“s in Zeiten der Cloud-Wirtschaft wurde das Konzept des „Distributed Digital Twin“s (dDTw) vorgeschlagen. Dieses fokussiert darauf, die Datenhoheit und -verantwortung, welche aus dem Entstehungsprozess resultiert, zu wahren. Relevant wird dann die Verknüpfung der verteilten

Artefakte, wie es auch bereits in den Konzepten MMC und ICDD vorgesehen ist. Semantische Technologien sind dafür gut geeignet, die technischen Grundlagen dazu werden in Abschnitt 2.3 erläutert. Eine vertiefte Einordnung der Lösungsansätze dieser Arbeit in das Konzept des dDTw erfolgt in Abschnitt 6.4.6.

1.2.3.2 BIM und KI

Gegenstand aktueller Forschungsarbeiten im Bereich BIM sind insb. die Schnittstellen zur künstlichen Intelligenz. Dabei stehen im Bereich Machine Learning generative Modelle für das Rendering und die Erstellung von Architekturentwürfen sowie die automatisierte Bildauswertung, z.B. im Bereich der Schadenserkennung und Vermessung, im Blickpunkt. Für die TGA-Planung bestehen derzeit Anknüpfungspunkte in der automatisierten Erkennung von Bestandsplänen und Bestandsaufnahmen wie z.B. 360°-Fotos und Punktfolgen.

Unterschiedliche Planungsbeteiligte bedienen sich in ihrer täglichen Arbeit häufig unterschiedlicher Fachbegriffe, beschreiben aber dennoch dieselben Dinge - dies kann für die modellbasierte Zusammenarbeit problematisch sein. Einen Lösungsansatz dafür bieten sog. *Ontologien*. Die Fachsprachen können in *Ontologien* abgebildet werden, *Alignments* dienen der Zuordnung der Begriffe unterschiedlicher Welten und können daher die notwendige Übersetzung zwischen den Fachsprachen abbilden. Details dazu werden in Abschnitt 2.3 behandelt.

Wissensgraphen und Ontologien sind Bestandteile von Expertensystemen. Diese erklärbaren KI-Systeme (Einordnung siehe Abb. 11) sind ebenfalls Gegenstand vieler Veröffentlichungen. In diesem Kontext wurden zahlreiche neue Ontologien entwickelt, welche im folgenden Abschnitt dargestellt werden. Lösungen zur Übersetzung zwischen Modelica und IFC mittels KI wurden jedoch noch nicht veröffentlicht.

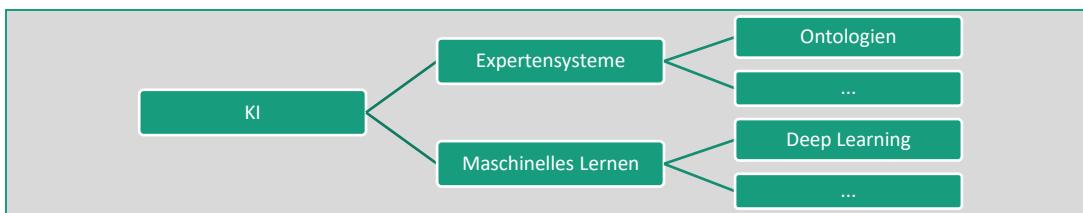


Abb. 11 Einordnung von Ontologien in Künstliche Intelligenz (KI)

1.2.3.3 Ontologien im Bereich Anlagentechnik, Regelungstechnik, Messdaten

Die zentrale Rolle von Funktionsmodellen wurden in Abschnitt 1.1.2 beschrieben. Es wurde eine Recherche zu möglichen herstellerneutralen Formaten der Darstellung dieser Funktionsmodelle durchgeführt. Viele Autoren schlagen die Verwendung von Wissensgraphen unter Verwendung von Ontologien dafür vor. Die wesentlichen Arbeiten in diesem Bereich werden im Folgenden zusammengefasst.

Für die Beschreibung von Anlagen ist oftmals der Bezug zur Gebäudetopologie relevant. Eine wesentliche Ontologie zur Beschreibung der Gebäudetopologie ist ifcOWL (Pauwels und Terkaj 2016), diese spiegelt den jeweils aktuellen Versionsstand der Industrial Foundation Classes (IFC) wieder. Da das IFC-Schema relativ komplex ist, wurde SimpleBIM (Pauwels und Roxin 2016) als vereinfachendes Postprocessing vorgeschlagen. Als Überontologie bzgl. der Gebäudetopologie wurde Building Topology Ontology (BOT) vorgeschlagen (Rasmussen u. a. 2017; 2017), dies wird von der Linked Building Group des W3C unterstützt.

Gebäudetopologie

IfcOWL ist ebenso für die Beschreibung von Anlagentechnik geeignet. (Teclaw u. a. 2023) zeigen beispielhaft an einer CaseStudy, dass damit bereits viele wesentliche Informationen ausgedrückt werden können.

Anlagen

1 Motivation, Stand von Wissenschaft und Technik

1.2 Status Quo BIM

1.2.3 Forschung

SAREF (Smart Appliances Reference Ontology) (Daniele, Hartog, und Roes 2015) dient der Beschreibung von Anlagen mit einem geringen Level of Detail (LOD). Es wurde durch (Villalón und Castro 2017) mit SAREF4BLDG unter Berücksichtigung der ISO 16739 (IFC) speziell auf die Gebäudedomäne angepasst. Die Haystack Tagging Ontology (HTO) (Charpenay u. a. 2015) beschäftigt sich mit semantischen Tags für Datenpunkte der Gebäudeautomation. (Bhattacharya, Ploennigs, und Culler 2015) verglichen 2015 die damals bereits vorhandenen Ontologien Haystack, IFC und SAREF in Hinblick auf die skalierbare Anwendung für das Gebäudeenergiemanagement. Sie erkannten Entwicklungsbedarf für Ontologien im Bereich der Expressivität (Sprachumfang) und der Praktikabilität. Die vorgeschlagenen Sensorontologien wurden als zu generisch bewertet, was ihre praktische Anwendung verhindert. Weitere Reviews zu verfügbaren Ontologien im Bereich der Anlagen- und Regelungstechnik sind enthalten in (Daniele, Hartog, und Roes 2015; Pauwels, Zhang, und Lee 2017; Rasmussen u. a. 2017; Schneider 2019b).

Mit der Flow System Ontology (FSO) (Kukkonen 2022) wurde eine weitere Ontologie speziell für die Gebäudeanlagentechnik vorgeschlagen. Diese wurde von (Pauen 2022; Pauen u. a. 2022) zur „TUBES system ontology“ ergänzt. Ihr Fokus liegt auf der Beschreibung von TGA-Anlagen und ihrer Beziehung zur räumlichen Struktur eines Gebäudes.

BRICK (Balaji u. a. 2018) ist ebenfalls eine Ontologie für die Beschreibung von physikalischer, logischer und virtueller Objekten in Gebäuden. Es wurde seither kontinuierlich weiterentwickelt und liegt mittlerweile als Version 1.4¹³ vor. BRICK behandelt sowohl die Anlagen- (HLK und Elektro), als auch die Gebäudeautomationstechnik. Die Entwicklung der Ontologie erfolgte durch ein akademisches Konsortium in den USA, es existiert inzwischen jedoch auch ein kommerzielles Konsortium, welches die Entwicklung unterstützt.

(Schneider 2019a; 2017) legte Zuordnungen von AEC-domänenspezifischen Ontologien (SAREF und BRICK) zur Gebäudetopologie mit BOT vor.

Gebäudeautomation

Im Bereich der Gebäudeautomation wurden zahlreiche Ontologien vorgeschlagen, viele davon wurden jedoch nicht weiter aufgegriffen: (Cyganiak 2012; Reinisch, Kofler, und Kastner 2010; Tomasevic u. a. 2015).

SSN (Compton u. a. 2012) und dessen Erweiterung SOSA (Haller u. a. 2018) sind Ontologien für Sensor- und Messdaten und wurden von zahlreichen Autoren aufgegriffen: (Kaiser und Stenzel 2015) binden es in die ESIM-Ontologie für das Energiemanagement von Anlagen ein. (Rasmussen u. a. 2018) erstellten ein Alignment zu BOT. (Pruvost 2022) schlägt ein Ontologiesystem für Sensorik in der Betriebsphase von Gebäuden vor, welches über SSN und SOSA hinaus weitere der o.g. Ontologien einbindet: ESIM, BOT, SAREF und BRICK. Auch (Terkaj, Schneider, und Pauwels 2017) folgen dem Grundprinzip des Ontologieentwurfs, möglichst wenig neu zu entwickeln, stattdessen existierende Module einzubinden. Sie erweitern SSN und SOSA um Module für die Gebäudeautomation, z.B. für Objektstatus und Performanceverlauf.

(Schneider, Pauwels, und Steiger 2017; Schneider 2019a; 2019b) schlagen mit CTRLOnt eine Ontologie speziell für die Abbildung von Controllerlogik vor, diese verwendet die von (Terkaj, Schneider, und Pauwels 2017) vorgeschlagene expression-Ontologie weiter. Als zentrale Klasse dient dabei der ControlActor, welcher mit seinen Inputs, Outputs und Parametern beschrieben wird. Außerdem kann die eigentliche Controllerlogik beschrieben werden, dafür sind z.B. Zeitpläne, Zweipunktregler, algebraische Ausdrücke vorgesehen. Ontologien für die Abbildung von

¹³ <https://docs.brickschema.org/intro.html>

Gebäudeautomationshardware sind BASont (Ploennigs u. a. 2012), welche von (Dibowski 2013) erweitert wurde.

1.3 Status Quo der Simulation in der Gebäudeenergiedomäne

1.3.1 Arten von Simulationsverfahren

In Abschnitt 1.1.1.1 wurden einige Beispiele für Simulationsverfahren in der Gebäudedomäne genannt. Diese können entsprechend ihres „Model of Computation“ (MoC) wie folgt gruppiert werden

1. Thermische Simulation („Gebäudesimulation“)
2. Thermohygrische Simulation (Bauteile)
3. Thermohydraulische Simulation (Heiz-, Kühl-, Lüftungsanlagen – „Anlagensimulation“)
4. Diskrete Modellierung: CFD „Computational Fluid Dynamics“, FEM

Verschiedene MoC können im Rahmen einer gekoppelten Simulation verbunden werden. In (Eckstädt u. a. 2020) wurden Simulationsszenarien hinsichtlich des Zusammenwirkens von Gebäude- und Anlagensimulation detailliert herausgearbeitet, Abb. 12 zeigt dazu eine Übersicht. Dargestellt sind von links nach rechts die jeweils zu berücksichtigenden Aspekte der Energieumwandlungskette im Gebäude, sowie eine Zuordnung zu Gebäudesimulation, Anlagensimulation oder Postprocessing. Unter Postprocessing werden in diesem Zusammenhang stationäre Berechnungen im Nachgang zur eigentlichen Simulation verstanden. Man erkennt, dass die Anlagensimulation i.d.R. gekoppelt zur Gebäudesimulation zu verwenden ist – eine Ausnahme bildet das Szenario III „Variantenstudie Anlagentechnik“, welches für die Erstellung des Anlagenkonzepts, wie in Abschnitt 1.1.2 bei der „Zentralplanung“ beschrieben, relevant ist. Dieses Simulationsszenario soll im Fokus dieser Arbeit stehen, da es erlaubt die spezifischen Herausforderungen der gekoppelten Simulation zunächst außen vor zu lassen. Das Modell kann in späteren Planungsphasen, denen die Simulationsszenarien III-VI zuzuordnen sind, sowie in der Betriebsphase und für die Planung der Gebäudeautomation nachverwendet werden. Dies wird im Ausblick (Abschnitten 6.4.1.1, 6.4.1.3 und 6.4.3) erläutert.

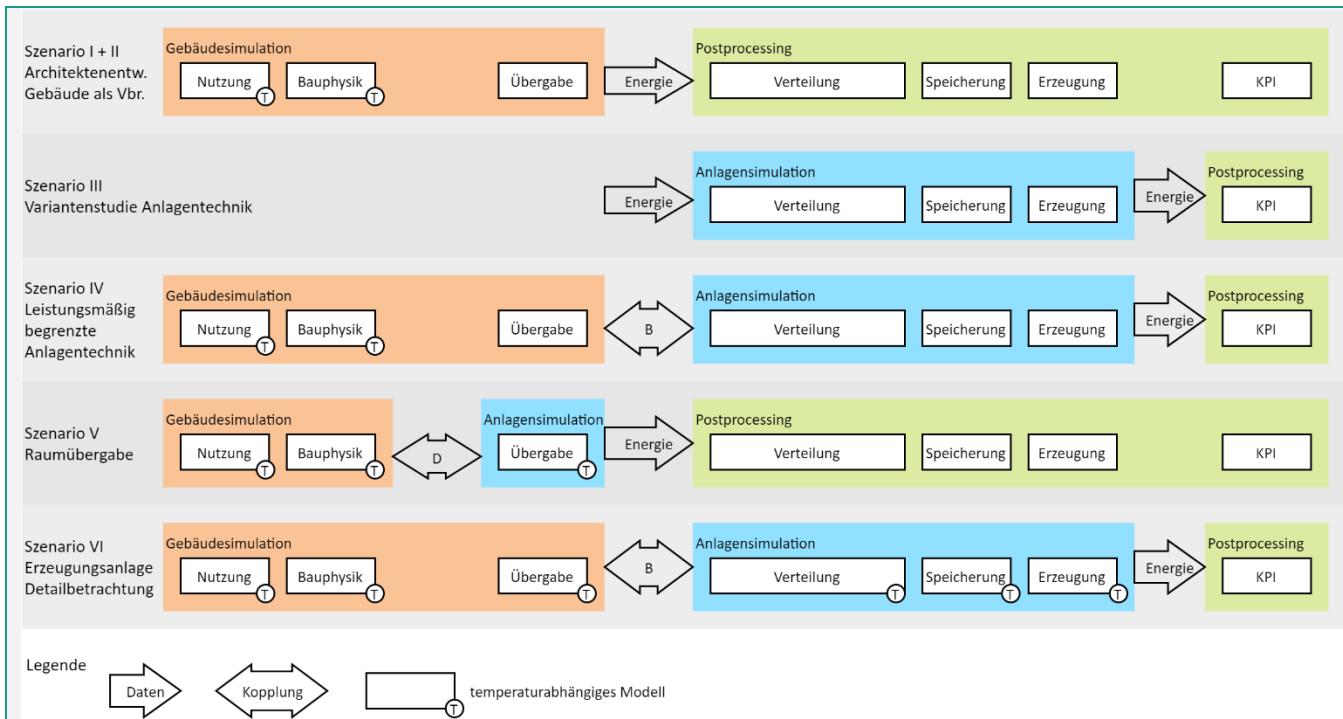


Abb. 12 Szenarien der Thermischen/Thermohydraulischen Simulation in der Planungsphase (BauSIM2020)

Je nachdem, welche konkrete Aufgabe im Planungsprozess gerade bearbeitet wird, existiert eine unterschiedlich enge Kopplung zwischen Objektplanung (Architekt) und Fachplanung TGA. Die Lastermittlung für die Auslegung der TGA als zeitiger Schritt im TGA-Planungsprozess zeichnet sich beispielsweise durch eine enge Kopplung zwischen Architektur und TGA aus. Hierfür kann - alternativ zu (quasi)statischen Berechnungsverfahren wie der Heizlastberechnung nach DIN EN 12831-1 - eine **Gebäudesimulation** (GS) angewendet werden, welche im Wesentlichen die thermischen Wechselwirkungen zwischen Wetter, Gebäudemassen und Innenräumen untersucht.

Für die Konzeption der Anlagentechnik, welche gewisse definierte Innenraumbedingungen gewährleisten soll, kann - alternativ zu klassischen Planungsverfahren wie z.B. VDI 4645 - eine **Anlagensimulation** (AS) zur Anwendung kommen. Dabei werden die Wechselwirkungen der anlagentechnischen Komponenten (Wärmepumpe, Speicher, BHKW, Netz, Verbrauch) untereinander untersucht. Im Bereich der Fachplanung TGA kommen vorrangig thermo-hydraulische Simulationen zum Einsatz, sie werden im Folgenden als „**Anlagensimulation**“ bezeichnet und bilden den Schwerpunkt dieser Arbeit. Die Gebäudesimulation liefert i.d.R. die Randbedingungen für die Anlagensimulation.

1.3.2 Simulationswerkzeuge

Simulationswerkzeuge können - wie oben beschrieben - entsprechend ihres Model of Computation kategorisiert werden, im Folgenden werden ausschließlich Werkzeuge für Gebäude- und Anlagensimulation (Nr. 1 und 2) betrachtet. Ein Simulationswerkzeug benötigt i.d.R. drei Bestandteile:

- Simulator (Simulationskerne)
- Benutzerschnittstelle (oftmals grafisch: GUI)
- Modellbibliothek

Manchmal sind mehrere Bestandteile in einem Werkzeug (sog. „Simulationsumgebung“) integriert, manchmal existieren die Bestandteile voneinander separiert. In Tab. 7 sind verschiedene Simulatoren, welche im Gebäudesektor für GS und AS verwendet werden, gegenübergestellt. Darunter sind universelle Simulatoren wie MATLAB und die Modelica-basierten Simulatoren, sowie Simulatoren, welche speziell für die Gebäudedomäne entwickelt wurden. Letztere haben Performancevorteile bei der Gebäudesimulation, für die Anlagensimulation ist ein spezialisierter Simulator nicht notwendig. Die Gründe dafür wurden in (Huang und Eckstädt 2023) dargestellt. EnergyPlus und NANDRAD sind Beispiele für nicht integrierte Simulatoren. Dazu existieren separate Benutzerschnittstellen, z.B. SIM-VICUS für NANDRAD und OpenStudio für EnergyPlus, welche ggf. eine abweichende Lizenzierung vom Simulator haben.

In der Tabelle ist weiterhin die vom Simulator genutzte Modellierungssprache angegeben. Wenn eine solche existiert, ist es für den Nutzer und Dritte möglich, eigene Modelle zu formulieren und somit die Modellbibliothek zu erweitern. Insb. für die Universalsimulatoren ist dies möglich, aber auch nötig, da die integrierten Modellbibliotheken kaum Funktionsumfang für die Gebäudedomäne bieten.

Tab. 7 Übersicht Simulatoren

	Simulator	Grafische Benutzerschnittstelle	Modellierungssprache * proprietär ** offen	Lizenzierung	Integrierte Modellbibliothek für Gebäudedomäne
universell	MATLAB	Grafisch, „Blockschaltbild“	MATLAB*	kommerziell	-
	Dymola	Grafisch, „Blockschaltbild“	Modelica**	kommerziell	MSL***
	Jmodelica	Grafisch, „Blockschaltbild“	Modelica**	kommerziell	MSL***
	OpenModelica	Grafisch, „Blockschaltbild“	Modelica**	FLOSS	MSL***
	SimulationX	Grafisch, „Blockschaltbild“	Spezielles Modelica-„Flavour“**	kommerziell	MSL***
Gebäude- und Anlagenfokus	TRNSYS	Grafisch, „Blockschaltbild“	TRNSYS	kommerziell	Umfangreich: GS+AS
	NANDRAD	3D Oberfläche für GS: SIM-VICUS	NANDRAD-Inputfile**	FLOSS	GS
	TRNSYS-TUD	grafische Oberfläche für Netzberechnung	TRNSYS-TUD-Inputfile* Erweiterbarkeit durch eigene Types in Hochsprachen (FORTRAN, Python, C++, Qt)	Nicht öffentlich verfügbar	GS+AS, sowie thermohygrische Simulation und Anbindung an CFD
	EnergyPlus	es existieren verschiedene tlw. kommerzielle GUI	IDF** Inputfile	FLOSS	umfangreich: GS+AS
	IDA ICE	Grafisch: 3D für GS, Schaltbild für AS	NMF**	kommerziell	umfangreich: GS+AS

***Modelica Standard Library hat einen sehr geringen Funktionsumfang für die Gebäudedomäne

Auch hinsichtlich der Simulationswerkzeuge, soll möglichst freie Software verwendet werden. Weitere wichtige Aspekte bei der Auswahl sind der Umfang der verfügbaren Modellbibliotheken für die Anlagentechnik und die Erweiterbarkeit der Modellbibliotheken. Dafür ist eine gut dokumentierte Modellierungssprache notwendig, idealerweise wird diese ergänzt von einer großen und aktiven Community. Tab. 7 und Tab. 8 zeigen, dass die Modelica-basierten Simulatoren die genannten Kriterien am besten erfüllt.

Tab. 7 zeigt, dass bei Modelica als einziger Modellierungssprache die theoretische Möglichkeit besteht, mit ein und demselben Modell zwischen offenen und kommerziellen Simulatoren zu wechseln. Tab. 8 gibt einen Überblick über verfügbare Bibliotheken für Modelica-Simulatoren. Trotz gemeinsamer Modellierungssprache sind diese aber nicht uneingeschränkt kompatibel – die entsprechenden Angaben sind daher in der Tabelle zu finden. Detaillierte Ausführungen zu diesen Bibliotheken finden sich in Abschnitt 2.2.6.

Tab. 8 Übersicht Modelica-Bibliotheken für die Gebäudedomäne

Lib	Aktuelle Version (März 2024)	Dymola	JModelica	OpenModelica	SimulationX	Lizenz	Zugehörige wiss. VÖ
AixLib* 1.3.2 (9.2.23)	<input checked="" type="checkbox"/>	k.A.	<input checked="" type="checkbox"/> ab v1.3.2	k.A.	BSD3	(Maier, Laura u. a. 2023)	
MBL* 10.0.0 (5.9.23)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ab v8.0.0	k.A.	BSD3	(Wetter u. a. 2014)	
IDEAS* 3.0.0 (3.5.22)	<input checked="" type="checkbox"/>	k.A.	<input checked="" type="checkbox"/>	k.A.	BSD3	(Jorissen u. a. 2018)	
BS* 2.0.0 (22.5.17)	<input checked="" type="checkbox"/>	k.A.	<input checked="" type="checkbox"/>	k.A.	BSD3	(Nytsch-Geusen u. a. 2012)	
VEPZO -	<input checked="" type="checkbox"/>	k.A.	k.A.	k.A.	k.A.	(Norrefeldt 2013)	
GreenCity	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	kostenpflichtig	-	

* gehört zu den sog. IPBSA-Bibliotheken, diese werden im Abschnitt 2.2.6 auf S. 82 vertieft thematisiert (Tab. 24)

1.3.3 Forschung

Optimierte Gebäude- und Anlagenentwürfe mittels Simulationsmethoden werden von vielen Autoren untersucht (Alwalidi, Kheybari, und Hoffmann 2022; Pawar, Boranian, und Lang 2022; Weiß, Tribulowski, und Hirth 2022; Eckstädt und Bräunig 2024). Für die Anwendung von mathematischen Optimierungsverfahren sind parametrische Modelle eine Grundvoraussetzung. Im Fokus steht dabei allerdings i.d.R. die Gebäudekubatur, weniger die Anlagentechnik.

Viele Veröffentlichungen beschäftigen sich mit Machine Learning für die Modellerstellung oder -kalibrierung (Elmers und Hollberg 2022; Frahm u. a. 2022; Benz, Geske, und Voelker 2022). Damit erfolgt eine Änderung von der klassischerweise üblichen White-Box-Modellierung in der Gebäudedomäne zur Grey- oder Black-Box-Modellierung.

Gegenstand aktueller Forschungen im Bereich der Gebäudesimulation ist die Betrachtung höherer Granularitätsebenen als Gebäude und Bauteile – namentlich Quartiere und Kommunen (Volkmer u. a. 2022; Hirsch und Paepcke 2022; Geiger u. a. 2022). Dafür spielt die Datenbereitstellung aus Geographischen Informationssystemen (GIS) eine zunehmende Rolle. Relevant ist bei dieser Betrachtung auch die Rückkopplung aus dem Gebäude auf Versorgungsinfrastrukturen insb. das Strom- und Fernwärmennetz.

Auch an Meta-Modellen und gekoppelten Simulationen (Huang und Eckstädt 2023; Paepcke, Leuschke, und Hoch 2022) wird derzeit geforscht.

Schwerpunkt dieser Arbeit bildet der simulationsbasierte Entwurf (SIM2BIM) bzw. umgekehrt die BIM-basierte Simulation (BIM2SIM) von Anlagentechnik. Letztere ist Gegenstand zahlreicher Veröffentlichungen, besonders im Bereich der Gebäudesimulation. Diese werden im folgenden Abschnitt 1.4.1 genauer analysiert.

1.4 Status Quo des Zusammenwirkens von BIM und Simulation

1.4.1 BIM2SIM

Die Erzeugung von Simulationsmodellen aus BIM-Modellen ist seit einigen Jahren Gegenstand der Forschung. Review-Paper für dieses Gebiet wurden vorgelegt von (Andriamamonjy, Saelens, und Klein 2019; Ciccozzi u. a. 2023; Di Biccari u. a. 2023; Gao, Koch, und Wu 2019; Kamel und Memari 2018). Dabei wurden die Simulationsarten Gebäude- und Anlagensimulation betrachtet. Diese Recherchen beschränkten sich BIM-seitig nicht auf IFC - mit gbXML existiert ein weiteres Format, welches im Bereich der Simulation einige Verbreitung gefunden hat. Auch simulationsseitig wurden verschiedene Modellierungssprachen, nicht nur Modelica, berücksichtigt. In Tab. 9 sind ausgewählte Arbeiten klassifiziert dargestellt. Man erkennt, dass die Anlagensimulation nur von wenigen Autoren berücksichtigt wird. In der Regel wird von den Autoren, wenn Gebäude- und Anlagensimulation betrachtet werden, ein separates Vorgehen für beide MoC beschrieben. Die Anlagensimulation wird dabei häufig nicht aus BIM übernommen bzw. ist der manuelle

Nacharbeitsaufwand enorm. Insbesondere tritt dieses Problem auf, wenn Anlagensimulationsmodelle erzeugt werden sollen, bevor die Geometrie definiert wurde. Wie in Abschnitt 1.1.2 beschrieben wurde, ist dies jedoch der logische Ablauf der Planung. In diesem Fall ist die Arbeitsweise SIM2BIM eine sinnvolle Alternative (siehe 1.4.2).

(Di Biccari u. a. 2023) haben eine umfangreiche strukturierte Literaturrecherche zur Interoperabilität zwischen BIM und Simulation durchgeführt, dabei wurden die Stichwörter „Building Energy Modelling“ (BEM) und „Building Performance Simulation“ (BPS) in die Recherche einbezogen, was als Oberbegriff für Gebäude- und Anlagensimulation zu verstehen ist. Sie unterteilen die veröffentlichten Arbeiten in drei Gruppen: zunächst die „aktuelle Ingenieurpraxis“, welche i.d.R. Autodesk-Produkte nutzt und insofern als closed BIM einzuordnen ist. Eine zweite Gruppe sind „Skripting“-Ansätze, welche entweder in das Authoring-Werkzeug als Plugin eingebaut oder über sog. Middleware-Tools realisiert werden. Dabei werden i.d.R. weitere Inputdateien – über die BIM und Simulationsdatei hinaus – einbezogen, welche keinem standardisierten Format folgen. Als dritte Gruppe bezeichnen sie Lösungsansätze, welche auf IFC basieren und sich dessen flexible Struktur zu Nutzen machen, um alle simulationsrelevanten Informationen zu speichern.

Tab. 9 wissenschaftliche Veröffentlichung im Bereich BIM2SIM

Quelle	Ziel	Veröffentlichungen	GS**	AS**
IFC	Modelica	Siehe Tab. 10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IFC	EnergyPlus	(Gao, Koch, und Wu 2019)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> 17
		(Bazjanac und Maile 2004; O’Sullivan und Keane 2005)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		In (Kamel und Memari 2018): 3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Ahn u. a. 2014)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Porsani u. a. 2021)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Giannakis 2015; 2019)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IFC	Sonstige	11 Veröffentlichungen in (Gao, Koch, und Wu 2019) Zielsysteme u.A. TRNSYS, PHPP, DOE2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		4 in (Kamel und Memari 2018) Zielsysteme u.A. DOEs, Cometh, Matlab	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		eeEmbedded: (Katranuschkov u. a. 2016): Zielsysteme u.A. TRNSYS-TUD, CFD-Tools	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Leimbach 2022)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Elisabeth Eckstädt, Claudia Liersch 2022)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
gbXML	Verschiedene	19 Veröffentlichungen in (Gao, Koch, und Wu 2019)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> 19
		(Kamel und Memari 2018)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Stratbäcker u. a. 2017): Vergleich gbXML und IFC, aber keine Betrachtung von Modelica als Zielsystem	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Nasyrov 2013; Nasyrov u. a. 2014)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Bracht, Melo, und Lamberts 2021)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CAD*	verschiedene	(Odeh und de Wilde 2020): Revit->DesignBuilder	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Jeong u. a. 2014): Revit->Modelica (nur Gebäudesimulation)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		(Elisabeth Eckstädt, Claudia Liersch 2022)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

*proprietäre Formate, ** Angabe zur Anzahl bei Review-Papern

In (Nasyrov 2013; Nasyrov u. a. 2014) wurden verschiedene Simulationstools als Zielsysteme untersucht. Bzgl. der Anlagentechnik wurden jedoch nur bruchstückhafte Lösungen gefunden – keines der Tools bietet eine praxistaugliche, halbwegs vollständige Implementierung.

Im Bereich der Verbindung von IFC und Modelica sind die relevanten Beiträge im Umfeld des Projekts “IEA IBC Annex 60” (2014-2017) (Wetter und van Treeck 2017) und dem Folgeprojekt “IBPSA Project 1” (2017-2022) („IBPSA Project 1“ 2021) geleistet worden. Die relevanten Akteure sind hierbei Lawrence Berkeley National Laboratory (LBNL), USA um Michael Wetter und die RWTH Aachen, Lehrstuhl Energieeffizientes Bauen (E3D) um Prof. Christoph van Treeck als Koordinatoren beider Projekte. Weitere sehr aktive Beitragende sind die KU Leuven, Belgien um Prof. Dirk Saelens

(Jorissen u. a. 2018) und die UdK Berlin um Prof. Nytsch-Geusen (Nytsch-Geusen u. a. 2012; Nytsch-Geusen 2019). In Deutschland lief 2013 bis 2018 im Rahmen des Programms EnOB (energieoptimiertes Bauen) das Projekt "EnEff-BIM Planung, Auslegung und Betriebsoptimierung von energieeffizienten Neu- und Bestandsbauten durch Modellierung und Simulation auf Basis von Bauwerkinformationsmodellen" (van Treeck, Müller, von Both, Nytsch-Geusen, Maile, Wimmer 2017; Stratbäcker, Mitterhofer, Benndorf, Dang, Rehault 2017) in welchem teilweise die gleichen Partner tätig waren. Im Zuge dieser Projekte wurden die in Tab. 10 zusammengefassten Toolchains für die Erzeugung von Simulations- aus BIM-Modellen vorgeschlagen. Diese Lösungsansätze behandeln den Bereich Anlagensimulation und zugehörige Regelungstechnik meist gar nicht, diese sollen jedoch im Fokus der vorliegenden Arbeit stehen.

An der RWTH Aachen wurden einige Arbeiten im Bereich der Anlagensimulation durchgeführt: (Cao u. a. 2014; Remmen u. a. 2015; van Treeck u. a. 2017; Wetter und van Treeck 2017; Reinhard Wimmer 2020; R. Wimmer u. a. 2014). Diese verwenden das Format Simmodel (O'Donnell u. a. 2011), um insb. die simulationsrelevanten Informationen zur Anlagentechnik zu speichern. Es wurde eine Benutzeroberfläche entwickelt, um notwendige Daten anzureichern. Simmodel ist ein simulatorunabhängiges Dateiformat. Die für die Verwendung des vorgestellten Workflows notwendigen Softwarekomponenten stehen nicht open source zur Verfügung.

Tab. 10 wissenschaftliche Veröffentlichungen zu IFC2Modelica

Veröffentlichungen	Ziel-Library	GS	AS	Bemerkung
(Wetter und van Treeck 2017)		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Übergeordneter Projektbericht zum Projekt „Annex 60“
„BIM (BIM- Ifc2Modelica)“ (Andriamamonjy 2018; 2018; Reynders, Andriamamonjy, Klein, Saelens 2017)	IDEAS	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> nur RLT	genaue Entsprechung von Modelica-Modell zu IFC-Entität muss da sein, Ergänzung fehlender Sachen aus IFC im Tool Ifc2Modelica, dieses ist aber nicht aufzufinden
„BIM2Modelica“ (Nytsch-Geusen 2019)	AixLib BuildingSystems	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	(CoTeTo basierend auf SimModel), Ausführung mit JModelica als FMU
(Stratbäcker, Mitterhofer, Benndorf, Dang, Rehault 2017)	VEPZZO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
(van Treeck u. a. 2017; Cao 2018; Reinhard Wimmer 2020; Cao u. a. 2014; Remmen u. a. 2015)	AixLib, BuildingSystems, MBL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	zwischenmodell SimModel, Schwerpunkt auch auf HVAC

1.4.2 SIM2BIM

Der „Rückweg“ von der Simulation zu BIM wird in der Literatur kaum behandelt, wie auch (Li u. a. 2022) in ihrer Recherche herausfanden. Eine Ausnahme bilden die Veröffentlichungen eben dieser Autoren: In (Li und Zhang 2021) wird eine Methodik für den bidirektionalen Austausch zwischen dem EnergyPlus-Format IDF und IFC für bauphysikalische Komponenten (Wände, Fenster, etc.) beschrieben, in (Li und Zhang 2023) wird die Anlagentechnik thematisiert. Andere Autoren beschäftigen sich bzgl. des Rückwegs von Simulation zu BIM ausschließlich mit der Übertragung von Eigenschaften als IfcProperties.

Wie in Abschnitt 1.1.2 aufgezeigt wurde, existiert insb. für den Entwurf von Anlagenkonzepten jedoch auch der Anwendungsfall des Erstentwurfs im Simulationstool – ein Anwendungsfall den auch (Li und Zhang 2023) adressieren. Das Funktionsmodell wird dann im Simulationswerkzeug erstellt. Dieses Funktionsmodell kann anschließend mit Geometrie angereichert werden, um daraus ein Konstruktionsmodell zu generieren. Um die Anreicherung zu ermöglichen, wird das Simulationsmodell zunächst in das IFC-Format überführt – dies wird in dieser Arbeit als UseCase2 (UC2) bezeichnet und in Abschnitt 1.5.1 vertieft beschrieben.

1.4.3 Ontologien für Modelica-Modelle

Die Technologien des Semantic Web (ausführliche Erläuterungen dazu finden sich in Abschnitt 2.3 ab S. 84) erfahren in den letzten Jahren eine zunehmende Verbreitung und scheinen ein

vielversprechender Ansatz für die Übersetzung zwischen SIM und BIM. Dabei spielen Ontologien eine entscheidende Rolle.

Ontologien für die Beschreibung von Katalogen von Simulationsmodellen wurden von (Mitterhofer 2018) mit „FMUOn“ und (Lee und Neuendorffer 2000) mit „MoML“ vorgeschlagen. (Delgoshaei, Austin, und Veronica 2017) beschreiben die Kopplung von MATLAB mit Dymola für die Anlagensimulation mittels einer Ontologie. Sie beschränkten sich allerdings auf das Speichern von Simulationsergebnissen in einem semantischen Format und eine konkrete Implementierung wurde nicht veröffentlicht.

Die Abbildung einer Modellierungssprache als Ontologie hilft dabei, Modelle aus dieser Sprache dauerhaft nutzbar und konvertierbar in andere Sprachen zu machen. Pop und Fritzson beschrieben bereits 2003 (Pop und Fritzson 2004; 2003), dass sowohl die Modelica Language Specification als auch die *Modelica Standard Library* als Ontologien verstanden werden können und zeigten sowohl die Vorteile dieser wissensbasierten Darstellung auf, als auch eine Repräsentation eines Beispielsmodells mit OWL-Vokabular. Die Arbeiten dazu wurden von den Autoren aber nicht weiter verfolgt. Im Projekt SPRINT (Shani u. a. 2014; Shani 2017) wurden die Vorteile der Abbildung einer Modellierungssprache als Ontologie erneut aufgezeigt. Für Modelica und SysML wurden OWL-Repräsentationen erzeugt und basierend darauf ein Übersetzer vorgestellt: *Semantic Mediation Container* (SMC). Ein ähnlicher Ansatz wird in dieser Arbeit mit dem *Semantic Translator* verfolgt. In diesem Zusammenhang wurde die *Wolfram System Modeler Ontology* (wsm) veröffentlicht, die in Abschnitt 3.6 beschrieben und weiterentwickelt wird.

Ein zweigeteiltes Vorgehen (d.h. eine Ontologie für die Fachlichkeit und eine für den Simulationsteil) wurde bei (Novak und Sindelar 2011; Silver, Hassan, und Miller 2007) beschrieben. (Novak und Sindelar 2011; Sindelar und Novak 2016) beschreiben die Erstellung von Simulationsmodellen für industrielle Anwendungen, basierend auf einem Wissensgraphen einer realen Fabrik und einer Ontologie der zugehörigen Simulationsbibliothek. Ein analoger Ansatz wird in dieser Arbeit für die TGA-Domäne mit der „Procedural Translation“ verfolgt.

Das Whitepaper von (Zeb und Kortelainen 2021) untersuchte eine semantische Darstellung von Modelica-Modellen als eine Option für langfristige Dateninteroperabilität. Sie schlussfolgerten, dass die RDF-Darstellung keine gute Lösung für langfristige Dateninteroperabilität ist, jedoch in dem Bewusstsein, dass lediglich ein simplifizierter Ansatz beschrieben wurde, ohne geeignete Domänenbibliotheken zu berücksichtigen. Die Veröffentlichung von (Roxin, Dundee, und Vukovic 2021) verfolgte denselben Ansatz, wobei sie selbst im Ausblick die Verwendung von Domänenbibliotheken als vielversprechenden Ansatz beschrieben haben. Dieser Ansatz wird in dieser Arbeit untersucht und in den Kapiteln 3 und 3.11 näher beschrieben.

(Nachawati u. a. 2022) beschreiben ein Framework für Systems Engineering mit Modelica namens "GitWorks". Dieses enthält einen Modellkatalog mit dem Namen "GitWork Commons", der einen Wissensgraphen der vorhandenen Modelle darstellt. Für deren Erzeugung beschreiben sie einen Parser, der die Modelle in semantische Darstellung umträgt. Als Ontologie verwenden sie die Eigenentwicklung "ModelicaOML". Sie wurde nicht mit OWL definiert, sondern mit der sog. "Ontological Modeling Language" OML, die im OpenCeasar-Projekt entwickelt wurde, aber kein W3C-Standard ist. Während die ModelicaOML auf Github¹⁴ veröffentlicht ist, ist der Transkriptor als Teil des GitWorks Frameworks nicht veröffentlicht.

¹⁴ <https://github.com/OpenModelica/ModelicaOML>

1.5 Forschungsbedarf, Ableitung der Aufgabenstellung

Im Bereich Simulation zur Unterstützung der Planungsprozesse für Gebäude gibt es – wie oben gezeigt – zahlreiche Vorarbeiten. Diese unterscheiden sich zunächst nach dem betrachteten Gewerk bzw. dem Model of Computation. Viele beschäftigen sich mit der Domäne der Bauingenieure und Architekten, die Bauphysik mit ihren vielen Facetten bietet zahlreiche Anwendungsfälle für Simulationsuntersuchungen, steht aber nicht im Fokus dieser Dissertation. Vielmehr sollen Beiträge in der Domäne der Fachplaner für technische Gebäudeausrüstung (TGA) entstehen, dabei vorrangig bezüglich der energieintensiven Gewerke Heizung, Lüftung, Kälte (HLK) sowie der zugehörigen Gebäudeautomation (GA) – mithin steht die Anlagensimulation im Fokus der Betrachtungen.

Bei der Übersetzung zwischen BIM- und Simulationsmodellen besteht Forschungsbedarf insb. in Hinblick auf die Anlagensimulation. Während bereits zahlreiche Lösungsansätze für die Überführung von Gebäudegeometrien aus BIM-Modellen in Simulationsmodelle existieren, gibt es im Bereich der Anlagentechnik bisher nur wenige Veröffentlichungen. Die dafür vorgestellten Lösungsansätze (Andriamamonjy, Saelens, und Klein 2018; Pinheiro u. a. 2018) basieren auf einer Ergänzung von für eine Simulation notwendigen Eingabedaten in einem dritten Format (neben dem BIM-Modell und dem Simulationsmodell) – einem Simulations-Meta-Modell. Damit einher geht ein zusätzlicher Einarbeitungsaufwand in die Werkzeuge zur Eingabedatengenerierung. Weiterhin müssen diese Werkzeuge sowie die Konvertertools verfügbar sein, was für die beschriebenen Lösungsansätze nicht gegeben ist.

In dieser Arbeit soll ein alternativer Ansatz untersucht werden: Ausschließlich simulationsrelevante Informationen sollen im Simulationswerkzeug eingepflegt werden. Informationen, welche für verschiedene Planungsbeteiligte interessant sind, sollen in unterschiedlichen Tools eingepflegt werden und anschließend miteinander geteilt werden können. Je nachdem, welcher Workflow in einem konkreten Projekt zur Anwendung kommt, können unterschiedliche Beteiligte (und die ihnen zugeordneten präferierten Softwarewerkzeuge) Autoren bzw. Urheber bestimmter Informationen sein. Daraus ergibt sich die Anforderung einer bidirektionalen Übersetzbartigkeit.

Der Übersetzungsprozess zwischen beiden Formaten soll modular und bidirektional aufgebaut werden. Es werden dabei auf Ontologien basierende Wissensgraphen als Zwischenergebnisse gewählt, um alternative Verwendungen dieser Zwischenergebnisse vorzubereiten. Ein besonderes Augenmerk soll neben der schon vielfach thematisierten Arbeitsweise „BIM2SIM“ auf einem simulationsbasierten Entwurf der Anlagentechnik und der dadurch notwendigen SIM2BIM-Übersetzung liegen.

Die SemanticWeb-Technologien (Wissensgraphen und Ontologien) erfahren in den letzten Jahren zunehmende Verbreitung und erscheinen als vielversprechender Lösungsansatz, um die Brücke zwischen BIM-Modellen und Simulationsverfahren zu schlagen. Das Simulationsmodell als Funktionsmodell kann damit Bestandteil des o.g. Multimodells werden – unter Wahrung der Zuständigkeiten und Urheberrechte der Planungsbeteiligten.

An die beteiligten Softwarewerkzeuge (BIM- und simulationsseitig) wird die Anforderung gestellt, dass sie die relevanten Informationen in ein offen definiertes Dateiformat exportieren können, konkret werden in dieser Arbeit die Formate IFC und Modelica¹⁵ untersucht. Für beide gibt es unterschiedliche Autorentools am Markt – von FOSS (kostenfrei) bis hin zu hochpreisigen kommerziellen Lösungen.

¹⁵ Die Auswahl dieser Formate wird in den Abschnitten 2.1 und 2.2 erläutert.

1.5.1 UseCases

Für diese Arbeit wird davon ausgegangen, dass das in Abschnitt 1.1.2 beschriebene Funktionsmodell ein Simulationsmodell ist, da dies wie o.g. viele Vorteile bietet. Daraus ergeben sich zwei UseCases:

UC1 BIM2SIM beschreibt die Erzeugung eines Simulationsmodells aus einem existierenden Konstruktionsmodell. Für die Anlagensimulation ist dieser vor allem für die „Nachrechnung“ bereits geplanter oder existierender Anlagen notwendig, z.B. für die Betriebsoptimierung oder Planung der Gebäudeautomation. In Abb. 6 und Abb. 8 entspricht dies der Nummer 1 bzw. der Nummer 3.

UC2 SIM2BIM beschreibt den Entwurf eines Anlagenkonzepts in der Simulationsumgebung (z.B. OpenModelica) und die anschließende Erzeugung eines IFC-Modells daraus. Damit erfolgt eine „Übersetzung“ in die Sprache bzw. ein Format der anderen Planungsbeteiligten außerhalb des Simulationskontextes. Sinnvoll ist dies z.B. für den Erstentwurf von Anlagenkonzepten. Für das Handling in IFC-Tools oder den Import in Autorentools muss i.d.R. eine 3D-Geometrie ergänzt werden, da die gegenwärtig verfügbaren Werkzeuge kein Handling auf Basis einer zweidimensionalen schematischen Darstellung unterstützen. In Abb. 6 und Abb. 8 entspricht dies der Nummer 2 bzw. der Nummer 4.

Detaillierte Anforderungen an die Übersetzung in beiden UseCases werden in Abschnitt 4.1 erarbeitet und dargestellt. Abb. 13 fasst die technologischen Kernbestandteile dieser Arbeit und die untersuchten UseCases zusammen.

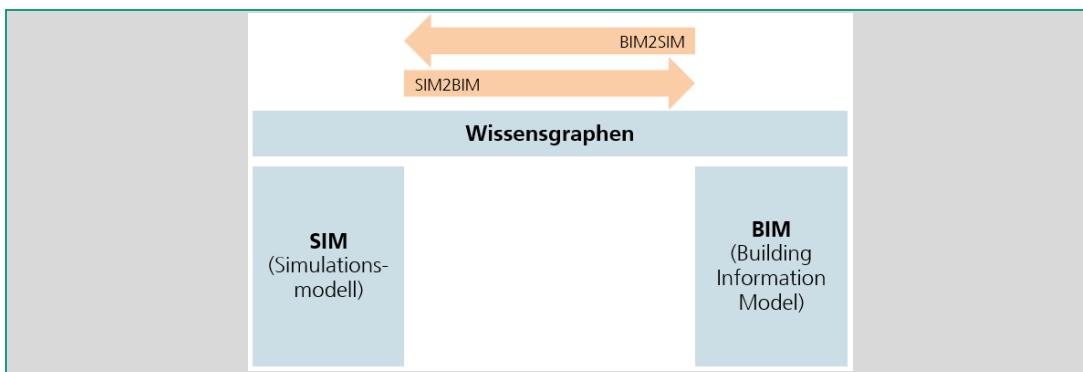


Abb. 13 Übersicht zu den technologischen Kernbestandteilen und den UseCases dieser Arbeit

1.5.2 Thesen

Es werden im Rahmen der Arbeit folgende Thesen geprüft:

1. Aus IFC-Modellen von Konstruktionen lassen sich automatisiert simulationsrelevante Informationen für die Anlagensimulation extrahieren.
Die dafür notwendigen Voraussetzungen sollen mit marktverfüglichen Softwarewerkzeugen erfüllbar sein. Der dafür nötige Aufwand wird abgeschätzt.
2. IFC ist ein offenes Format um Funktionsmodelle abzuspeichern, aus denen Schalt- und Strangschemata (als wichtiges Werkzeug der Planung) abgeleitet werden können.
Eine zweidimensionale schematische Darstellung kann zusätzlich zur 3D-Geometrie in IFC abgelegt werden. Dies gewährleistet die Synchronität der Informationen zwischen zweidimensionaler Schemadarstellung und dreidimensionalem Konstruktionsmodell.
3. Aus einem solchen „IFC Schaltschema“ lassen sich Modelica-Modelle erzeugen, ohne dass weiteren Informationsquellen einbezogen werden müssen.

Die entstandenen Modelica-Modelle sollen über eine valide Modellstruktur und eine geeignete Vorparametrierung verfügen, um sie mit geringem Aufwand in einen rechenfähigen Zustand bringen zu können.

Als Zwischenschritt für die Übersetzung werden Wissensgraphen genutzt.

Der mit der Übersetzung einhergehende Informationsverlust wird dokumentiert.

4. Die in These 3 beschriebe Übersetzung funktioniert bidirektional.

Die Abbildung von Modelica-Modellen als Wissensgraph ist möglich und geeignet als Ausgangspunkt für eine Übertragung von Informationen in ein IFC-Modell.

Die entstehenden IFC-Modelle sollen in marktverfügaren BIM-Tools weiterbearbeitet werden können, um einen simulationsbasierten Anlagenentwurf zu unterstützen.

Der mit der Übersetzung einhergehende Informationsverlust wird dokumentiert.

1.6 Gliederung dieser Arbeit

Die weitere Arbeit unterliegt daher folgender Gliederung:

- In Kapitel 2 werden die methodischen und technischen Grundlagen für die konkret gewählten Modellierungssprachen von BIM (IFC) und Simulation (Modelica) dargestellt. Außerdem werden die Grundlagen der semantischen Technologien (semantic web technologies SWT) mit ihren wesentlichen Bestandteilen Wissensgraphen und Ontologien erläutert.
- In Kapitel 3 wird der methodische Ansatz für die Kopplung von BIM und SIM mittels semantischer Technologien erläutert.
- In Kapitel 4 wird die Implementierung erläutert. Dabei werden zunächst Anforderungen an die Implementierung formuliert. Basierend darauf wurde eine modulare Architektur bestehend aus neun Konverterkomponenten abgeleitet. Die Konverterkomponenten werden in drei Gruppen (Pre- und Postprocessing, Transkriptions- und Translationskomponenten) unterteilt. Alle Konverter werden im Einzelnen erläutert.
- In Kapitel 5 wird der implementierte Lösungsansatz anhand der beiden UseCases BIM2SIM und SIM2BIM experimentell getestet. Der Vor- und Nachbereitungsaufwand wird dokumentiert und die Anwendung der einzelnen Konverter demonstriert.
- In Kapitel 6 werden die Erkenntnisse aus Implementierung und Experiment zusammengefasst. Es erfolgt eine Prüfung der Thesen. Es wird dargestellt, für welche weiteren UseCases die implementierten Konverter anwendbar sind sowie welche weiteren Forschungsfragen sich für nachfolgende Arbeiten ergeben.

2 Methodische und Technische Grundlagen

In diesem Kapitel werden zunächst die wesentlichen technologischen Grundlagen und Möglichkeiten des openBIM-Formats IFC dargestellt. Dazu gehören die Informationen zum Dateiformat IFC-STEP, als auch die relevanten Konzepte ausgewählter IFC-Datenschemata.

Anschließend wird ein Überblick über die Simulationsumgebung Modelica gegeben. Es wird die Sprachspezifikation erläutert, ein Überblick über verfügbare Simulationsumgebungen gegeben und ausgewählte Modellbibliotheken erläutert.

Abschließend werden Grundlagen der Semantischen Technologien erläutert. Dazu gehören Wissensgraphen, Ontologien und ihre Einordnung in den Bereich der künstlichen Intelligenz. Es werden marktverfügbare Werkzeuge zum Umgang mit Wissensgraphen und Ontologien dargestellt. Außerdem werden verfügbare Ontologien für die Domäne Anlagen- und Regelungstechnik dargestellt.

2.1 IFC¹⁶

2.1.1 Allgemeines

Alle in dieser Arbeit erarbeiteten BIM-Workflows haben den Anspruch, das Open-BIM-Konzept zu erfüllen, d.h. herstellerneutral anwendbar zu sein.

In den 2000er Jahren wurden die Industrial Foundation Classes (IFC) als Datenschema vom heutigen Verein buildingSMART vorgeschlagen und 2005 erstmals als ISO 16739:2005 normiert. Branchenweit verbreitet ist inzwischen Version IFC2x3 (veröffentlicht 2005). Mit IFC4 erfolgte 2013 eine wesentliche Erweiterung des Datenschemas, welche seit 2018 in der Version 4.0.2.1 als ISO 16739-1:2018 normiert ist. IFC 4.3 wurde kürzlich in der Version 4.3.2.0 als ISO 16739-1:2024 normiert, IFC 4.4 ist Gegenstand aktueller Vornormierungsarbeit im buildingSMART Verband. Alle folgenden Ausführungen basieren auf dem IFC 4.0.2.1, wenn nicht anders angeben.

2.1.2 IFC-Datenschemata

Übersicht

Die Industrial Foundation Classes definieren die im Folgenden kurz eingeführten 38 Datenschemata, die sich den vier konzeptionellen Ebenen zuordnen lassen:

- Ressourcenebene / *resource layer*: 21 Datenschemata
- Kernebene / *core layer*: 4 Datenschemata
- Interoperabilitätsebene / *interop layer*: 5 Datenschemata
- Fachbereichsebene / *domain layer*: 8 Datenschemata

In jedem Datenschema können *Types*, *Entities*, *PropertySets*, *QuantitySets*, *Functions* und *Rules* definiert werden, eine Übersicht zeigt Tab. 11. *Types*, *Entities*, *Functions* und *Rules* beginnen mit dem Präfix *Ifc*, im Schema vordefinierte *PropertySets* beginnen mit dem Präfix *Pset_*¹⁷, *QuantitySets* mit dem Präfix *Qto_*. Danach erfolgt eine Fortsetzung mit englischen Bezeichnern in der CamelCase-Namenskonvention, die daher auch in dieser Arbeit verwendet wird.

Entities sind dabei mit dem Konzept von *Klassen* in anderen Programmiersprachen vergleichbar, jedoch enthalten *Entities* in IFC nur eine Datenstruktur, keine Methoden. *Entities* haben in der Regel etwa zehn Attribute, die im IFC-Schema definiert sind.

resource layer

Der *resource layer* spielt eine besondere Rolle: Die Datenschemata der Ressourcenebene stellen unterstützende Datenstrukturen bereit. Entitäten und Typen, die in dieser Schicht definiert sind, können von allen Entitäten in den anderen Schichten referenziert werden. Im Gegensatz zu Entitäten in anderen Layern können Datenstrukturen aus dem *resource layer* (**Ressourcenentitäten**) nicht eigenständig existieren, sondern nur, wenn sie (direkt oder indirekt) von einer oder mehreren Entitäten referenziert werden, die von IfcRoot abgeleitet sind. Da Ressourcendefinitionen kein Identitätskonzept (wie z. B. eine GUID) haben, implizieren mehrere Objekte, die auf dieselbe Instanz einer Ressourcenentität verweisen, keine Beziehung. Beispielsweise sind zwei Polylinien (*IfcPolyline*), die sich dieselbe Instanz für einen Punkt (*IfcCartesianPoint*) teilen, und zwei Polylinien, die unterschiedliche Instanzen für identische Punkte verwenden, semantisch gleichwertig. Es wird empfohlen (ist aber nicht zwingend), dass

¹⁶ Die Abschnitte 2.1.2 bis 2.1.9 wurde von der Autorin verfasst und in ähnlicher Form bereits im Abschlussbericht FMI4BIM (Eckstädt, Huang, u. a. 2023) veröffentlicht.

¹⁷ Dieser Präfix ist exklusiv reserviert für im Schema vordefinierte PropertySets

Anwendungen die Dateigröße minimieren, indem sie nach Möglichkeit identische Ressourcenentitäten verwenden.¹⁸

Tab. 11 Mächtigkeit der 38 Datenschemata in IFC 4.0.2.1

Datenschema	Types	Entitie s	Property Sets	Quantity Sets	Function s	Rule s
5. Core data schemas						
5.1 IfcKernel	10	50	1		5	1
5.2 IfcControlExtension	1	3				
5.3 IfcProcessExtension	8	11	1			
5.4 IfcProductExtension	14	57	29	6		
6. Shared element data schemas						
6.1 IfcSharedBldgElements	23	52	24	16		
6.2 IfcSharedBldgServiceElements	4	26	41	1		
6.3 IfcSharedComponentElements	4	10	14			
6.4 IfcSharedFacilitiesElements	4	28	28			
6.5 IfcSharedMgmtElements	5	5	11			
7. Domain specific data schemas						
7.1 IfcArchitectureDomain	9	7	64			
7.2 IfcBuildingControlsDomain	6	12	1	6		
7.3 IfcConstructionMgmtDomain	6	14	139	3		
7.4 IfcElectricalDomain	22	44	202	22		
7.5 IfcHvacDomain	33	66	50	31		
7.6 IfcPlumbingFireProtectionDomain	5	10	1	5		
7.7 IfcStructuralAnalysisDomain	11	28	14			
7.8 IfcStructuralElementsDomain	10	17		3		
8. Resource definition data schemas						
8.1 IfcActorResource	3	8				
8.2 IfcApprovalResource		3				
8.3 IfcConstraintResource	5	5				
8.4 IfcCostResource	2	3				
8.5 IfcDateTimeResource	13	14				
8.6 IfcExternalReferenceResource	9	11				
8.7 IfcGeometricConstraintResource	5	11				
8.8 IfcGeometricModelResource	4	42		1		
8.9 IfcGeometryResource	14	59		2		
8.10 IfcMaterialResource	4	18	14	25		
8.11 IfcMeasureResource	112	11		1		
8.12 IfcPresentationAppearanceResource	25	44		4		
8.13 IfcPresentationDefinitionResource	2	6		1		
8.14 IfcPresentationOrganizationResource	4	10				
8.15 IfcProfileResource	4	27	3			
8.16 IfcPropertyResource	2	14		1		
8.17 IfcQuantityResource		9		1		
8.18 IfcRepresentationResource	4	17		8	1	
8.19 IfcStructuralLoadResource	6	20				
8.20 IfcTopologyResource	1	20		3		
8.21 IfcUtilityResource	3	5				
Total	397	776	420	93	47	2

Im IFC-Schema werden neben *occurrence* auch *object type* Entities definiert (z.B. *IfcSpace* und *IfcSpaceType*). Diese werden im Folgenden als **Instanz-Objekte** bzw. **Type-Objekte** bezeichnet.

Instanz- und Type-
Objekte

Letztere können instanziert werden, um als Datencontainer für gemeinsame Daten mehrerer Objekte zu dienen. Die Gruppe *Types* enthält die Definition von Datentypen, darunter primitive

¹⁸ Absatz ist eine fast wörtliche Übersetzung aus der IFC-Spezifikation
https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/link/chapter-8.htm

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.3 IFC-STEP-Dateien

Datentypen wie *IfcReal*, *IfcLabel* und zahlreiche *Enumerations>Selects*. Tab. 12 zeigt ausgewählte *Entities* und *Types*, welche für die weitere Verwendung in dieser Arbeit wichtig sind.

Tab. 12 Ausgewählte Types und Entities aus ausgewählten Datenschemata

Datenschema	Types	Entities
5 Core data schemas		
5.1 IfcKernel	IfcPropertySetDefinitionSet IfcObjectTypeEnum	IfcRoot IfcRelDefines IfcPropertySet IfcProject IfcProduct
5.2 IfcControlExtension		IfcPerformanceHistory IfcRelAssociatesConstraint
5.4 IfcProductExtension	IfcSpaceTypeEnum	IfcBuilding IfcElement IfcRelConnectsPorts IfcPort IfcSite IfcSpace IfcSpaceType IfcSystem IfcZone
6 Shared element data schemas		
6.1 IfcSharedBldgElements	IfcWallTypeEnum	IfcSlab IfcWall IfcDoor
6.2 IfcSharedBldgServiceElements	IfcDistributionSytsemEnum	IfcFlowSegment IfcDistributionPort IfcRelFlowControlsElements
7 Domain specific data schemas		
7.2 IfcBuildingControlsDomain		IfcController IfcSensor
7.4 IfcElectricalDomain		IfcLamp IfcJunctionBox
7.5 IfcHvacDomain	IfcBoilerType IfcFanType IfcHumidifierType ...	IfcBoiler IfcFan IfcHumidifier IfcChiller ...
8 Resource definition data schemas		
8.11 IfcMeasureResource	IfcValue IfcUnit IfcText IfcReal IfcLabel IfcIdentifier IfcBoolean	IfcSIUnit IfcDerivedUnit

Schemadefinitionen

Das IFC-Schema wurde vom Verein buildingSMART als sog. EXPRESS-Schema spezifiziert und wird außerdem als XML-Spezifikation (XSD) und als OWL-Schema veröffentlicht.

2.1.3 IFC-STEP-Dateien

Neben Datenschemata, welche in den folgenden Abschnitten näher erläutert werden, legen die IFC auch zwei Strukturen für Austauschdateiformate fest: XML und STEP. Das STEP-Format nach ISO 10303-21 ist deutlich verbreiteter und wird auch in dieser Arbeit (für die Implementierung und auch die Codelistings) genutzt. Es handelt sie bei diesen Dateien um ASCII-Dateien. In der Datensektion sind die eigentlichen Inhalte entsprechend des IFC-Schemas (zugehörig ist hier die EXPRESS-Darstellung des Schemas) enthalten.

Bei einer IFC-STEP-Datei handelt es sich um eine Liste von Instanzen, die jeweils einen innerhalb der Datei eindeutigen Namen haben. Der Instanzname (im Folgenden auch Instanz-ID) hat die Form

einer positiven Ganzzahl mit einem vorangestellten Doppelkreuz. Anschließend folgt der Klassename in Großbuchstaben und anschließend in Klammern die zugeordneten Attribute. Diese können auch im Verweis auf eine andere Instanz bestehen. Ein Beispiel zeigt Lis. 1.

Nicht gesetzte Attribute werden durch ein Dollarzeichen gekennzeichnet. Enumerationen und logische Werte werden ebenfalls in Großbuchstaben mit vor- und nachgestelltem Punkt dargestellt.

```
#100=IFCPROJECT('0xScRe4drECQ4DMSqUjd6d',#110,'proxy with CSG',$$,$,$,$,(#201),#301);
#110=IFCOWNERHISTORY(#111,#115,$,.ADDED.,1320688800,$,$,1320688800);
#111=IFCPERSONANDORGANIZATION(#112,#113,$);
#112=IFCPERSON($,'Liebich','Thomas',$,$,$,$,$);
#113=IFCORGANIZATION($,'buildingSMART International',$,$,$,$);
#115=IFCAPPLICATION(#113,'1.0','IFC text editor','ifcTE');
#201=IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,1.0E-5,#210,$);
#202=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',,,,#201,$,.MODEL_VIEW.,$);
#210=IFCAxis2Placement3D(#901,$,$);
#301=IFCUNITASSIGNMENT((#311,#312));
#311=IFCSIUNIT(,.LENGTHUNIT,.MILLI.,.METRE.);
#312=IFCCONVERSIONBASEDUNIT(#313,.PLANEANGLEUNIT.,'degree',#314);
#313=IFCDIMENSIONALEXPONENTS(0,0,0,0,0,0,0);
#314=IFCMEASUREWITHUNIT(IFCPLANEANGLEMEASURE(0.017453293),#315);
#315=IFCSIUNIT(,.PLANEANGLEUNIT,$,.RADIAN.);
#500=IFCBUILDING('2FCZDorxHDT8NI01kdXi8P',$,'Test Building',$,$,#511,$,$,.ELEMENT.,$,$,$);

```

Lis. 1 Ausschnitt aus der DATA-Sektion (DATA section) einer IFC-STEP-Datei¹⁹

2.1.4 Wesentliche abstrakte Entitäten zur Datenverwaltung

2.1.4.1 IfcRoot

Das IFC-Datenmodell enthält zahlreiche Klassen. Der gemeinsame Supertyp aller Entitäten, außer derer, die auf der Ressourcenebene definiert wurden, ist die abstrakte Klasse *IfcRoot* aus dem Datenschema *IfcKernel*. Alle Entitäten, die Subtypen von *IfcRoot* sind, können selbstständig verwendet werden, während Entitäten aus der Ressourcenebene nicht selbstständig existieren können. *IfcRoot* hat folgende Attribute²⁰

- #1 GlobalId
- #2 OwnerHistory
- #3 Name
- #4 Description

Außer der *GlobalId* sind alle Attribute optional. Weitere Ausführungen zur Verwendung von *GlobalId* und *OwnerHistory* finden sich in Abschnitt 2.1.8. Beispiele für *IfcRoot*-Objekte (mit *GlobalID*) und Entitäten aus der Ressourcenebene sind in Lis. 1 zu sehen. Die *GlobalId* ist ein sog. *Globally Unique Identifier* (GUID). Sie bildet den eindeutigen Identifikator für *IfcRoot*-Objekte, der im Gegensatz zur Instanz-ID in der STEP-Repräsentation auch über Dateigrenzen hinweg eindeutig ist. Es handelt sich dabei i.d.R. um eine automatisch generierte 128Bit-Zahl.

2.1.4.2 IfcObject

Die abstrakte Entität *IfcObject* definiert das Attribut

- #5 ObjectType.

¹⁹ <https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2/HTML/link/csg-primitive.htm>

²⁰ Die Nummerierung der Attribute mit vorangestelltem Doppelkreuz wird in der IFC-Dokumentation benutzt und daher hier so übernommen.

welches einer Klassifikation der Instanzen dient, wenn im Attribut *PredefinedType* die Option *.USERDEFINED* ausgewählt wurde, andernfalls erfolgt die Klassifikation durch die Zuordnung eines Type-Objekts mittels der Relation *IfcRelDefinesByType*. Details dazu sind in Abschnitt 2.1.5 dargestellt.

Subtypen von *IfcObject* sind die wesentlichen Informationscontainer in IFC-Modellen, welche andere Informationscontainer enthalten oder Referenzieren können. Dazu dienen die Relationsobjekte *IfcRelDefinesByType*, *IfcRelDefinesByObject* und *IfcRelDefinesByProperties*, welche sich auf Instanzen von *IfcObject* beziehen.

2.1.4.3 *IfcProduct*

Eine weitere wesentliche Klasse ist die abstrakte Entität *IfcProduct*, welche die Attribute

- #6 ObjectPlacement
- #7 Representation

definiert. Sie dient damit zur Darstellung aller Objekte, welche eine geometrische oder räumliche Zuordnung haben. Die Geometrie des Objekts wird durch *Representation* beschrieben, dabei können jedem Objekt mehrere Repräsentationen zugeordnet werden, beispielsweise eine 2-dimensionale und eine 3-dimensionale. Die Platzierung im Koordinatensystem kann relativ zu lokalen Koordinatensystemen oder absolut im Globalkoordinatensystem erfolgen.

Abb. 14 zeigt die Subklassen von *IfcProduct*.

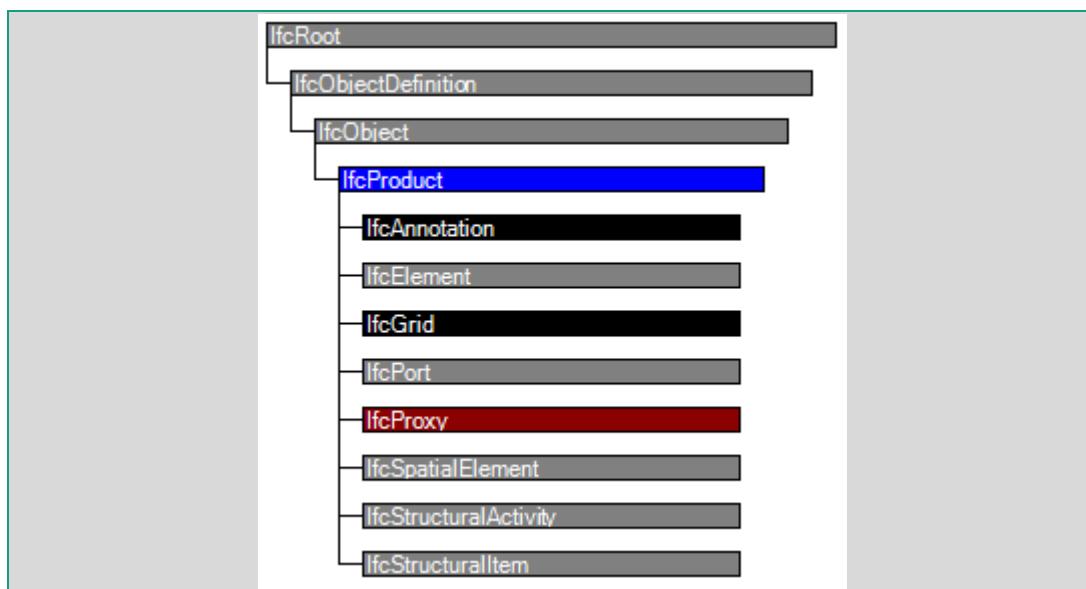


Abb. 14 Klassenhierarchie von *IfcProduct*

IfcAnnotation ist gemäß der Schemadefinition für Texte, Vermessungspunkte und Ähnliches vorgesehen. Das IFC-Schema sieht damit eine Möglichkeit vor, derartige Objekte als eigene Entitäten vom Typ *IfcAnnotation* ins Modell aufzunehmen, statt als Anmerkung zu den zugehörigen Entitäten.

Die Entität *IfcProxy* ist seit IFC4 *deprecated*.

2.1.4.4 IfcElement

Unter *IfcElement* werden alle Dinge verstanden, die physisch existieren, darunter auch sog. *void*-Objekte wie z.B. Fensteröffnungen. Dazu gehören die Gebäudeelemente *IfcBuildingElement*, ebenso wie die TGA-Elemente *IfcDistributionElement* und weitere, siehe dazu Abschnitt 2.1.5.

Das *IfcVirtualElement* ist für die Bereitstellung imaginärer Begrenzungen vorgesehen. Die praktische Anwendung erfolgt derzeit vorrangig für sog. „Raumbegrenzungslinien“ zwischen zwei nicht durch ein Bauteil getrennten Räumen z.B. Wohn- und Essbereich in vielen Wohngrundrissen.

2.1.5 Entitäten mit physikalischer Entsprechung

2.1.5.1 Bauteile

Für die Abbildung von Hochbau-Bauteilen gibt es die abstrakte Klasse *IfcBuildingElement*. Die zugehörigen Subklassen sind in Abb. 15 dargestellt und umfassen alle wesentlichen für die Gebäudesimulation relevanten Bauteile wie Wände, Türen, Fenster, Decken und Dächer.

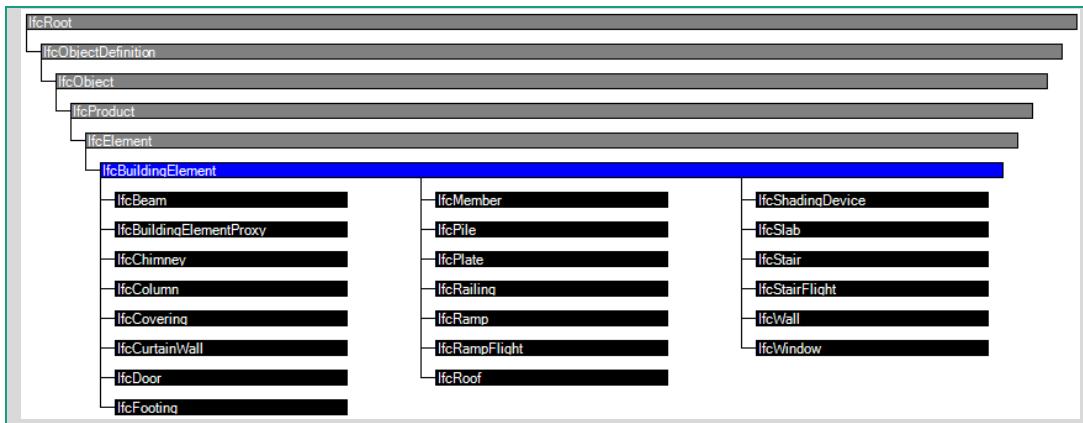


Abb. 15 Bauteile - definierte Entitäten aus dem Schema *IfcProductExtension*

2.1.5.2 Anlagentechnik

Anlagentechnik wird mit Entitäten der Klasse *IfcDistributionElement* dargestellt. Dafür können sowohl Subklassen, als auch die Klasse selbst genutzt werden. Sie dient für Anlagen aller Art, darunter Heizung, Kälte, Lüftung, Sanitär, Elektro, Gebäudeautomation. Abb. 16 zeigt die im IFC-Schema definierten Subklassen. Während die Klasse *IfcDistributionFlowElement* die mechanischen Komponenten definiert, enthält die Klasse *IfcDistributionControlElement* die zugehörigen Komponenten für die Gebäudeautomation. Der Begriff *Distribution* beschreibt in diesem Fall, abweichend zu der Begrifflichkeit *Verteilung* entsprechend der Energieumwandlungskette (wie sie z.B. in Abb. 12 dargestellt ist) alle Komponenten der Energieumwandlungskette von Erzeugung bis Übergabe.

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.5 Entitäten mit physikalischer Entsprechung

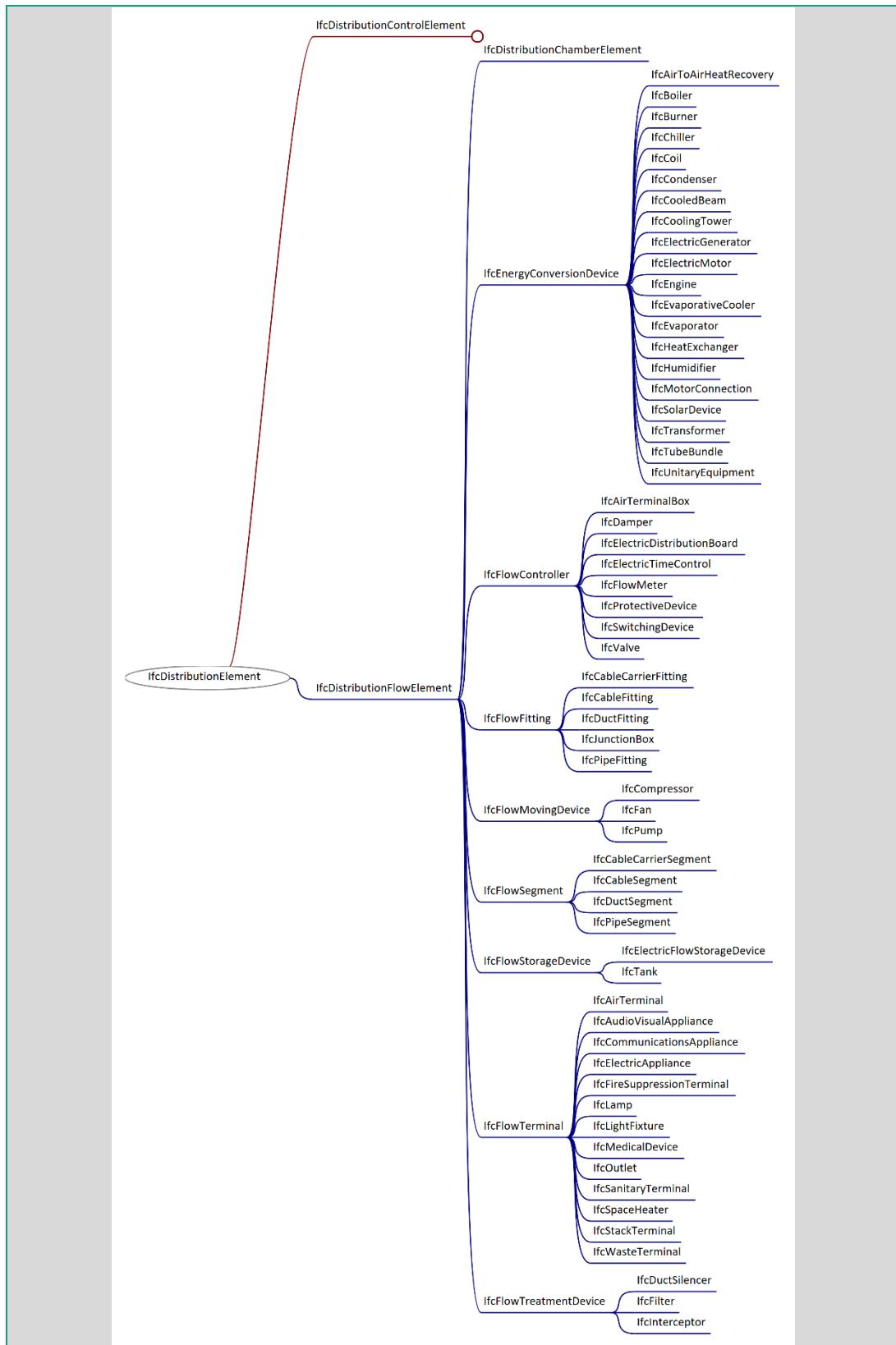


Abb. 16 definierte Anlagentechnische Elemente aus dem Schema IfcProductExtension

2.1.6 Typisierung und Klassifizierung von Objekten

Viele Entities (z.B. *IfcSpace* und *IfcSpaceType*) haben ein Attribut namens *PredefinedType*, dessen Datentyp eine zugehörige Aufzählung (*Enum* z.B. *IfcSpaceTypeEnum*) ist. Diese Aufzählungen sind innerhalb der Datenschemata in der Kategorie *Types* definiert. Innerhalb dieser Aufzählung kommen in der Regel *.USERDEFINED.* und *.NOTDEFINED.* vor. Wird ein Objekt durch eine *IfcRelDefinedByType*-Relation spezifiziert, darf das Attribut *PredefinedType* am Objekt nicht gesetzt sein. Das Aufzählungselement *.NOTDEFINED.* wird in der Regel für Type-Objekte wie z.B. *IfcSpaceType* genutzt, jedoch ist dies in der Spezifikation nicht so vorgegeben. Stattdessen könnte hier eine mehrstufige Klassifikation erfolgen, indem die Type-Objekte zusätzlich einen der vordefinierten Typen aus dem IFC-Schema erhalten.

Lis. 2 stellt drei Varianten ein Objekt zu klassifizieren gegenüber: Instanz #2 nutzt lediglich die vorgegebene Klassifizierung als *Space* (entspricht Innenraum), Instanz #3 bekommt einen *ObjectType* lediglich als Label zugeordnet, Instanz #4 bekommt ein *IfcSpaceType*-Objekt zugeordnet. Dieses kann Träger weiterer Eigenschaften oder Bestandteil weiterer Relationen sein.

```
#2=IFCSPACE('3T4wScwjfCkxuC4i0p4dUP',#48,'0.1/U.1',$,,$,#266,#2919,'Buerol',.ELEMENT., .SPACE.,$);
#3=IFCSPACE('3T4wScwjfCkxuC4i0p4dUP',#48,'0.1/U.1',$,,'Buero 2 Personen', #266,#2919,'Buerol',.ELEMENT.,.USERDEFINED.,$);
#4=IFCSPACE('3T4wScwjfCkxuC4i0p4dUP',#48,'0.1/U.1',$,,$,#266,#2919,'Buerol',.ELEMENT.,$,,$);
#5= IFCRELDEFINESBYTYPE('3bQUn0hwj8ax23Sw3cxBcu',#48,$,$,(#4),#6);
#6= IFCSpacETYPE('3SC5ohcI1Ag8MFJL7u086r',#48,'Buero 2 Personen',$$,$,$,$,'489748',$,.NOTDEFINED.,$);
```

Lis. 2 Drei Varianten einen Raum innerhalb von IFC zu klassifizieren

2.1.7 Zuordnung von Eigenschaften zu Objekten

Die im Schema definierten Entitäten haben i.d.R. bis zu 15 zugeordnete Attribute, welche sie von ihren Supertypen erben oder selbst definieren. Tab. 13 zeigt dies am Beispiel des *IfcSpace*. Man kann erkennen, dass es sich bei den vordefinierten Attributen im Wesentlichen um Metadaten und die Geometrie handelt. Weitere alphanumerische Informationen, welche für einen solchen Raum wichtig sind, können mit Attributen nicht gespeichert werden. Dies erfolgt über die Zuordnung von *PropertySets* oder *QuantitySets* zu Instanz- oder Type-Objekten. Im letzterem Fall besteht die Zuordnung zu Instanz-Objekten indirekt über das Type-Objekt. *PropertySets* und *QuantitySets* enthalten jeweils ein oder mehrere *Properties/Quantities*. Weitere Informationen können Entitäten implizit zugeordnet werden, indem sie zu verschiedenen Systemen und Gruppen zugeordnet werden. Dies wird in Abschnitt 2.1.9 beschrieben.

Tab. 13 Attribute von *IFCSPACE*

Nummer des Attributs	Attributname	Definiert bei
#1	GlobalId	IfcRoot
#2	OwnerHistory	IfcRoot
#3	Name	IfcRoot
#4	Description	IfcRoot
#5	ObjectType	IfcObject
#6	ObjectPlacement	IfcProduct
#7	Representation	IfcProduct
#8	LongName	IfcSpatialElement
#9	CompositionType	IfcSpatialStructureElement
#10	PredefinedType	IfcSpace
#11	ElevationWithFlooring	IfcSpace

Im IfcSchema existieren vordefinierte *Property* und *QuantitySets* für viele Entitäten, darunter das „Common“-PropertySet und die „BaseQuantities“ welche für die meisten Entitäten vordefiniert sind z.B. für Räume *Pset_SpaceCommon* und *Qto_SpaceBaseQuantities*.

Es ist möglich, darüber hinaus eigene *Property*- und *QuantitySets* zu erstellen. Das Template-Konzept erlaubt es projektspezifische Platzhalter für *Property*- und *QuantitySets* zu definieren. Für diese kann festgelegt werden, ob sie für Type- oder Instanz-Objekte zuordenbar sind oder ggf. für beides, wobei eine direkte Zuordnung vom Instanz-Objekt eine ggf. zusätzlich bestehende Zuordnung vom Type-Objekt überschreibt. *PropertySets* werden über die *IfcRelDefinesByTemplate*-Relation dem *PropertySetTemplate* zugeordnet. Diese Templates werden einer *ProjectLibrary* zugeordnet, welche nicht in der Modelldatei enthalten sein muss, sondern extern referenziert und damit separat versioniert werden kann.

PropertySets sind Zusammenstellungen verschiedener Eigenschaften, die Instanz- und Type-Objekten zugeordnet werden können. *QuantitySets* sind Zusammenstellungen von Quantitäten. Die Konzepte *Property* und *Quantity* sind in IFC eng verwandt: beide können in der Form *Simple* oder *Complex* auftreten. Bei letzterem enthalten sie mehrere *Simple* Komponenten. Sowohl *Property*, als auch *Quantity* haben die Attribute *Name*, *Description* und *Unit*, sowie den Wert an sich. Für *Quantities* sind fünf Spezielle Klassen vordefiniert (*Area*, *Count*, *Length*, *Time*, *Volume*, *Weight*), welche genutzt werden müssen, da der Supertype *IfcPhysicalSimpleQuantity* abstrakt ist. Soll eine andere Größe repräsentiert werden, muss dafür ein *IfcPropertySingleValue* genutzt werden, der dort anzugebende Wert ist vom Datentyp *IfcValue*, welcher sehr flexibel ist z.B. *IfcInteger*, *IfcBoolean*, *IfcLabel*, *IfcAreaMeasure*. Somit ist die Nutzung von *PropertySets* flexibler als die von *QuantitySets*. Jedoch kann durch Verwendung der speziellen Datentypen z.B. *IfcAreaMeasure* statt *IfcReal* dieselbe Expressivität erreicht werden. Für *Quantities* kann im Gegensatz zu *Properties* optional eine *Formula* angegeben werden, dabei kann es sich um eine mathematische Berechnung, Datenbanklinks oder Kombinationen derselben handeln. Sowohl für *Properties*, wie auch für *Quantities* können *Templates* definiert werden.

In Hinblick auf die Verbindlichkeit der Angabe der Eigenschaften kann folgende Reihenfolge gebildet werden:

1. Pflichtattribute,
2. optionale Attribute,
3. vordefinierte *PropertySets*,
4. frei definierte *PropertySets*.

2.1.8 Versionierung mittels IFC-Konstrukten

Das Änderungsmanagement spielt im Planungsprozess eine zentrale Rolle. Dies wurde in Abschnitt 4.2 des FMI4BIM-Abschlussberichts (Eckstädt, Huang, u. a. 2023) detailliert herausgearbeitet. Die technischen Grundlagen, um dies im IFC-Standard zu realisieren werden im Folgenden erläutert.

2.1.8.1 GlobalID

Das Konzept der GlobalID dient dem eindeutigen Identifizieren von Objekten, auch dann, wenn dies über andere Eigenschaften eines Objekts nicht mehr möglich wäre. Somit sollte die Entscheidung, ob eine GlobalID beibehalten wird, vom Nutzer sehr bewusst getroffen werden. Da der Nutzer i.d.R. nicht selbst die IFCs manipuliert, muss die Benutzeroberfläche des Programms diese Implikation beim anschließenden Erzeugen der IFC-Files eindeutig aufzeigen.

2.1.8.2 IfcOwnerHistory

In der IFC-Definition gibt es die Klasse *IfcOwnerHistory*, diese wird im Folgenden als **Änderungsgeschichte** bezeichnet. Eine Änderungsgeschichte wird jedem Objekt, welches von *IfcRoot* abgeleitet ist, als zweites Attribut zugeordnet (siehe z.B. in Tab. 13). Dazu gehören beispielsweise alle *IfcProduct* (d.h. alles was in Realität eine Geometrie hat), alle *IfcPropertyDefinition* und alle *IfcRelationship*.

Objekte vom Typ *IfcOwnerHistory* haben folgende Attribute:

Tab. 14 Attribute von *IfcOwnerHistory*

Nummer des Attributs	Attributname	Type
#1	OwningUser	IfcPersonAndOrganization
#2	Owning Application	IfcApplication
#3	State	IfcStateEnum
#4	ChangeAction	IfcChangeActionEnum
#5	LastModifiedDate	IfcTimeStamp
#6	LastModifyingUser	IfcPersonAndOrganization
#7	LastModifyingApplicationCreationDate	IfcApplication
#8	CreationDate	IfcTimeStamp

Demnach hat jedes *IfcRoot*-Objekt eine besitzende Person/Organisation und Software-Applikation, sowie ein Erstelldatum. Weiterhin hat es einen Status (les- und schreibbar, lesbar, weder noch). Außerdem enthält es Informationen zur letzten Änderung des Objekts: Autor, Zeitpunkt, Applikation und Art der Änderung.

```
IfcChangeActionEnum
NOCHANGE
MODIFIED
ADDED
DELETED
NOTDEFINED
```

Lis. 3 IfcChangeActionEnum

2.1.8.3 Darstellung eines idealtypischen Änderungsworkflows mit Hilfe der *IfcOwnerHistory*

Wird ein Modell erstmals vom Modellautor (Architekt) z.B. im Rahmen des UC1 erstellt, haben alle *IfcRoot*-Objekte eine Änderungsgeschichte, wie beispielhaft als Instanz #100 in Lis. 4 dargestellt.

```
#3 = IFCPERSONANDORGANIZATION(#4, #5, #6);
#4 = IFCPERSON($, 'Hentschel', , 'Alexander', $, $, $, $, $, $);
#5 = IFCORGANIZATION($, 'EA Systems Dresden', $, $, $);
#6 = IFCACTORROLE(.ARCHITECT., $, $)

#7 = IFCPERSONANDORGANIZATION(#8, #9, #10);
#8 = IFCPERSON($, 'Eckstädt', , 'Elisabeth', $, $, $, $, $, $);
#9 = IFCORGANIZATION($, 'Fraunhofer IIS/EAS', $, $, $);
#10 = IFCACTORROLE(.CONSULTANT., $, $)

#11 = IFCPERSONANDORGANIZATION(#12, #13, #14);
#12 = IFCPERSON($, 'Hoch', , 'Rene', $, $, $, $, $, $);
#13 = IFCORGANIZATION($, 'TU Dresden IBK', $, $, $);
#14 = IFCACTORROLE(.USERDEFINED., 'Ingenieur für Gebaeudesimulation', $)

#20 = IFCAPPLICATION(#21, '2020', 'Revit', 'Not Defined');
#21 = IFCORGANIZATION($, 'Autodesk', $, $, $);
#30 = IFCAPPLICATION(#13, 'Build 4711', 'SIMVICUS', 'Not Defined');

#100 = IFCOWNERHISTORY(#3, #20, .READONLY., .ADDED., 1614688338, #3, 20, 1614688338);

#1001 = IFCOWNERHISTORY(#3, #20, .READONLY., .NOCHANGE., 1614689000, #11, #30, 16146883381614689000);

#1002 = IFCOWNERHISTORY(#3, #20, $, .MODIFIED., $, $, $, 1614688338);
```

Lis. 4 Beispielhafte Entitäten für die Erstellung einer OwnerHistory

Ein ausschließliches Prozessieren in einem anderen Tool z.B. SIMVICUS würde eine Änderungsgeschichte #1001 erzeugen. Dabei sind Attribute *ChangeAction*, *LastModifiedDate*, *LastModifyingUser* und *LastModifyingApplication* gegenüber der vorher beschriebenen Änderungsgeschichte #100 geändert. *ChangeAction=NOCHANGE* dient der expliziten Angabe, dass keine Änderungen am Objekt vorgenommen wurden, im Gegensatz zu einer fehlenden Angabe.,

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.8 Versionierung mittels IFC-Konstrukten

welche sich ergibt, wenn das fragliche IFC-File neueren Datums ist, als der letzte Zeitstempel in der *OwnerHistory* eines Objekts.²¹

Gemäß der *IfcChangeActionEnum* sind für *IfcRoot*-Objekte grundsätzlich folgende Operationen möglich: Keine Änderung, Änderung, Hinzufügen, Entfernen (siehe Lis. 3). Die Definition dieser Operationen ist nur im Vergleich zu einem Referenzmodell sinnvoll, i.d.R. wird damit das Eingangsmodell der Operation gemeint sein. Eine Stelle, an der angegeben werden muss, auf welches Modell sich die Änderungsmarker beziehen, ist jedoch im IFC-Standard nicht definiert.

Die Operation „Entfernen“ ist dabei von untergeordneter Bedeutung, da ein entfernte Objekt nach der Operation i.d.R. im IFC-File nicht mehr enthalten sein wird, somit die Zuordnung einer entsprechenden Änderungsgeschichte auch nicht mehr möglich ist. Es wäre allerdings denkbar das *IfcRoot*-Objekt beizubehalten und lediglich mit dem *DELETE*-Flag zu versehen.

Wichtige Operationen sind somit Änderung *MODIFIED* und Ergänzung *ADDED*. Eine Ergänzung ist dabei dadurch gekennzeichnet, dass ein Objekt mit einer neuen *GlobalId* im IFC-Modell erscheint, somit sind die Attribute *OwningUser*, *OwningApplication* und *CreationDate* in diesem Fall immer übereinstimmend mit *LastModifyingUser*, *LastModifyingApplication* und *LastModifiedDate*, wie beispielhaft an Instanz #100 in Lis. 4.

Als *MODIFIED* soll in dem Zusammenhang ein Objekt gelten, von dem mindestens ein Attribut²² (außer der *GlobalId*) verändert wurde. Dies wäre bei einer Wand beispielsweise auch die geometrische Repräsentation. Ändert sich hingegen auch die *GlobalId*, so handelt es sich um zwei Objekte (das erste wurde entfernt und danach das zweite hinzugefügt). Diese Unterscheidung sollte sehr bewusst getroffen werden (siehe vorheriger Abschnitt).

Tab. 15 Attribute von *IfcWall*

Nummer des Attributs	Attributname	Definiert bei
#1	GlobalId	IfcRoot
#2	OwnerHistory	IfcRoot
#3	Name	IfcRoot
#4	Description	IfcRoot
#5	ObjectType	IfcObject
#6	ObjectPlacement	IfcProduct
#7	Representation	IfcProduct
#8	Tag	IfcElement
#9	PredefinedType	IfcWall

Eine Änderung von Geometrie (*Representation*) und/oder Platzierung (*ObjectPlacement*) führen somit in jedem Fall zum Status *MODIFIED*. *ObjectPlacement* und *Representation* sind keine Subklassen von *IfcRoot* und können somit keinen eigenen Änderungsmarker tragen. Ändert sich ihr Inhalt, muss der Änderungsmarker von der ändernden Anwendung bei allen referenzierenden *IfcRoot*-Objekten gesetzt werden (siehe Beispielinstanz #1002 in Lis. 4).

Ändern sich hingegen die zugeordneten *Properties*, oder *QuantitySets*, *Materialien* stellt dies keine Änderung des Objekts dar, da sowohl die Relationsobjekte, als auch die *PropertySets* und *QuantitySets* eigene Änderungsgeschichte haben und Änderungen davon somit auf diese Art und Weise gekennzeichnet werden sollten.

²¹ Generell gilt: die Instanz-IDs z.B. im Beispiel #101 sind zwischen verschiedenen IFC-Files nicht verbunden. Eine gleiche Nummer ist kein Hinweis auf ein gleiches Objekt, eine unterschiedliche Nummer auch kein Hinweis auf ein unterschiedliches Objekt. Auch sind sie nicht zur Wiedererkennung geeignet.

²² „Attribute“ wird hier bewusst in Abgrenzung zu „Properties“ verwendet.

2.1.9 Strukturierung mittels IFC

2.1.9.1 Anlagenstrukturen in IFC

Anlagen dienen der Gruppierung von realen anlagentechnischen Komponenten, wie sie im Abschnitt 2.1.5.2 beschrieben wurden. Die Anlagen haben keine physikalische Entsprechung, bilden vielmehr eine virtuelle Entität. Die geeignete Klasse im IFC-Schema wurde mit IFC4 eingeführt: *IfcDistributionSystem*. Ein *IfcDistributionSystem* ist ein spezielles System, welches dem Handling eines Verteilmediums dient z.B. Heizungswasser.

Die übergeordnete Klasse *IfcSystem* (siehe Abb. 17) ist laut IFC4-Spezifikation eine „organisierte Zusammenstellung zusammengehörender Teile“, die einem gemeinsamen Zweck dienen oder gemeinsam eine Funktion erfüllen. Ein System ist im Wesentlichen eine funktional zusammenhängende Menge von Gegenständen (*IfcProduct*).

Die Gruppierungsrelation zwischen dem System oder seinen Bestandteilen (*IfcDistributionElement*) wird mit *IfcRelAssignsToGroup* abgebildet.

Systeme können ineinander verschachtelt sein, dazu dient die *IfcRelAggregates*-Relation, beispielsweise kann ein Lüftungsstrang Subsystem einer speziellen Lüftungsanlage sein, beides würde mit der IFC-Klasse *IfcSystem* abgebildet.

Die Zuordnung vom System zu seinen versorgten räumlichen Bereichen (z.B. Räume, Geschosse, Bereiche, siehe Abschnitt 2.1.9) erfolgt mit der *IfcRelReferencesInSpatialStructure*-Relation (Die Relation *IfcRelServicesBuildings*, welche dafür in IFC 4.0.2.1 noch vorgesehen war, ist in IFC 4.3.2.0 als *deprecated* gekennzeichnet.).

Im IFC-Standard ist eine lange Liste von *PredefinedTypes* in der *IfcDistributionSystemEnum* enthalten, darunter mechanische Systeme wie *AIRCONDITIONING*, *CHILLEDWATER*, *CONDENSERWATER*, *DOMESTICCOLDWATER*, *HEATING*, *REFRIGERATION*, *VENTILATION*, aber auch elektrische Systeme wie *LIGHTING*, *MONITORINGSYSTEM*.

Verbindungen zwischen Komponenten einer Anlage werden mit Hilfe der abstrakten *IfcPort* bzw. *IfcDistributionPort* abgebildet. Diese sind mittels *IfcRelNests*-Relationen zu *IfcProducts* zugeordnet und mit *IfcRelConnectsPorts*-Relationen untereinander verbunden. Die Struktur aus Komponenten einer Anlage und ihren Verbindungen wird als **Anlagengraph** bezeichnet.

2.1.9.2 Räumliche Gebäudestrukturen und *IfcSpace*

Im Gegensatz zu den physischen Elementen (siehe *IfcElement*) sind Objekte der Klasse *IfcSpatialElement* nicht physisch existent, sondern dienen nur der Verwaltung und Gruppierung von Informationen.

Das IFC-Schema unterscheidet zwischen zwei Gruppen räumlicher Strukturelemente:

- *IfcSpatialStructureElement*
- *IfcSpatialZone*

Erstere beschreiben eine hierarchische räumliche Struktur, während letztere auch potenziell überlappende Elemente enthalten können (in diesem Fall ist die Summe aller dieser Volumina größer als das Gesamtvolumen des Gebäudes). Es wird zunächst die hierarchische räumliche Struktur beschrieben.

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.9 Strukturierung mittels IFC

Tab. 16 Attribute von IfcSpatialStructureElement

Nummer des Attributs	Attributname	Definiert bei
#1	GlobalId	IfcRoot
#2	OwnerHistory	IfcRoot
#3	Name	IfcRoot
#4	Description	IfcRoot
#5	ObjectType	IfcObject
#6	ObjectPlacement	IfcProduct
#7	Representation	IfcProduct
#8	LongName	IfcSpatialElement
#9	CompositionType	IfcSpatialStructureElement

Laut Definition im IFC-Standard ist ein Raum (*IfcSpace*) eine Fläche oder ein Volumen, welches durch (reelle oder virtuelle) Umfassungsflächen begrenzt ist und eine bestimmte Funktion erfüllt (ähnlich lautet die Definition in VDI 6070). Durch die „bestimmte Funktion“ ergeben sich in der Regel auch gemeinsame Eigenschaften der Flächen oder Volumina die zu einem Raum gehören. Ein *IfcSpace* ist der Teil der hierarchischen räumlichen Struktur eines Gebäudes, die sich aus jeweils beliebig vielen Ebenen der Entitäten

- Liegenschaft *IfcSite*,
- Gebäude *IfcBuilding*,
- Geschoss *IfcBuildingStorey*,
- Raum *IfcSpace*

zusammensetzt. Die Reihenfolge der Klassen muss dabei eingehalten werden, Lücken sind aber möglich. Jede dieser Entitäten kann als *COMPLEX*, *ELEMENT* oder *PARTIAL* ausgeprägt sein, Tab. 17 zeigt ein minimales Beispiel, Tab. 18 ein komplexes Beispiel. Diese Gliederungsstruktur ist der Struktur in VDI 6070 sehr ähnlich.

Die Beziehungen zwischen den Ebenen der Gebäudestruktur werden mit Hilfe der *IfcRelAggregates*-Relation ausgedrückt. Es handelt sich dabei um eine 1-n-Beziehung, d.h. jedes Objekt einer detaillierteren Ebene ist genau einem Objekt in der übergeordneten Ebene zugeordnet. Der entstehende Graph, darf nicht zyklisch sein.

Tab. 17 Beispiel für einfache räumliche Struktur in IFC

IfcSpatialStructureElement	CompositionType	Name (einzigartig)	Bemerkung
<i>IfcSite</i>	ELEMENT	Flurstück 234/3344	
<i>IfcBuilding</i>	ELEMENT	Bürogebäude	
<i>IfcBuildingStorey</i>	ELEMENT	UG1	
<i>IfcSpace</i>	ELEMENT	Lüftungszentrale	

Tab. 18 Beispiel für komplexe räumliche Struktur in IFC

IfcSpatialStructureElement	CompositionType*	Name (eindeutig)	Bemerkung
IfcSite A site is a defined area of land, possibly covered with water, on which the project construction is to be completed. A site may be used to erect, retrofit or turn down building(s), or for other construction related developments.	COMPLEX	Gewerbegebiet Süd	
	ELEMENT	Flurstück 234/3344	
	PARTIAL	Baufeld 1	
	PARTIAL	Baufeld 1 Teil 1	
IfcBuilding A building represents a structure that provides shelter for its occupants or contents and stands in one place	COMPLEX	Neubau medizinisches Zentrum	
	ELEMENT	Bürogebäude	
	PARTIAL	Nordturm	
IfcBuildingStorey The building storey has an elevation and typically represents a (nearly) horizontal aggregation of spaces that are vertically bound	COMPLEX	Untergeschosse	
	ELEMENT	UG1	
	PARTIAL	Nordbereich	Bereich mit abgesenktem Fußboden
IfcSpace	COMPLEX	Technikkern	
	COMPLEX	HLK	
	ELEMENT**	Lüftungszentrale	
	PARTIAL	Aufstellfläche RLT1	

* COMPLEX und PARTIAL können ineinander verschachtelt werden

** View definitions and implementation agreements may restrict spaces with CompositionType=ELEMENT to be non-overlapping

Neben der vordefinierten hierarchischen Gebäudestruktur mit *IfcSpatialStructureElement* gibt es im IFC Schema die Möglichkeit weitere räumliche Untergliederungen vorzunehmen. Dazu sind die Entitäten *IfcSpatialZone* und *IfcZone* geeignet, welche für sich im Gegensatz zu den *IfcSpatialStructureElements* überlappen dürfen. Abb. 17 zeigt (blau hervorgehoben) wie sich diese in die IFC-Klassenhierarchie einordnen. Daraus resultiert auch der wesentliche Unterschied zwischen Beiden: Während eine *IfcSpatialZone* eine eigene Repräsentation und Platzierung aufweisen kann (geerbt von *IfcProduct*) ist dies für *IfcZone* nicht möglich. Möchte man vorhandene Entitäten lediglich gruppieren ist *IfcZone* die geeignete Entität, da ohne eigene Geometrie auch keine Widersprüche entstehen können. Die Zuordnung erfolgt mit der *IfcRelAssignsToGroup*-Relation, während die Zuordnung zu einer *IfcSpatialZone* mittel *IfcRelContainedInSpatialStructure* erfolgen würden. *IfcSpatialZone* sind gemäß Standard für die Modellierung räumlich zusammenhängender Zonen wie z.B. Brand- und Bauabschnitte vorgesehen. Sowohl *IfcZone* als auch *IfcSpatialZone* können beliebig ineinander verschachtelt sein, dementsprechend ist es damit möglich, Räume (und im Falle der *IfcZone* auch andere Entitäten) nach verschiedenen Kriterien zu gruppieren z.B. entsprechend ihrer Nutzungsart, der Zuordnung zu einer Mieteinheit oder zu einer versorgenden Anlage - Abb. 18 zeigt ein Beispiel.

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.9 Strukturierung mittels IFC

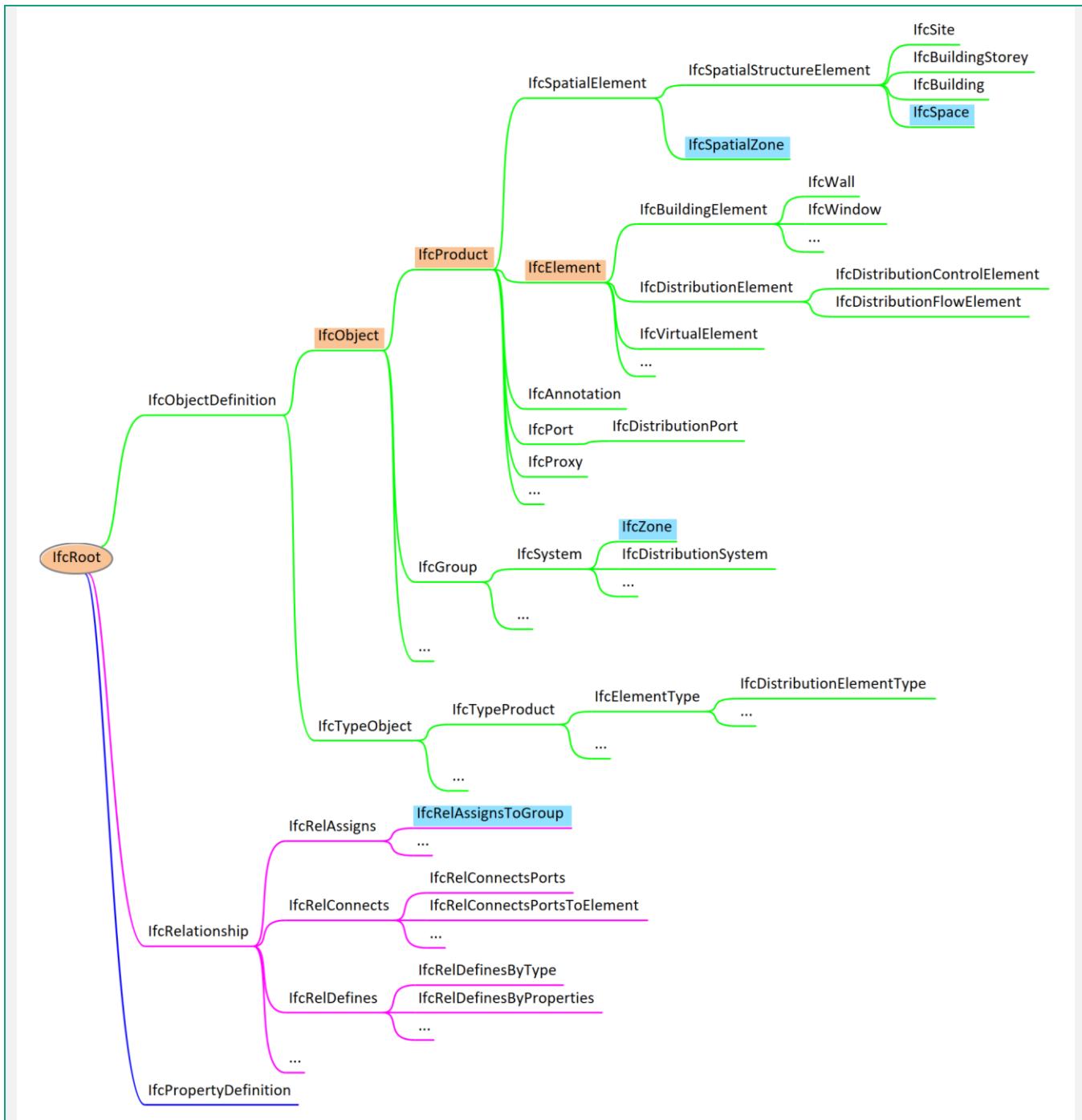


Abb. 17 ausgewählte Entitäten in der IFC-Klassenhierarchie
 (Erläuterung zu orangenen Entitäten siehe Abschnitt 2.1.4, zu blau siehe Abschnitt 2.1.9.2)

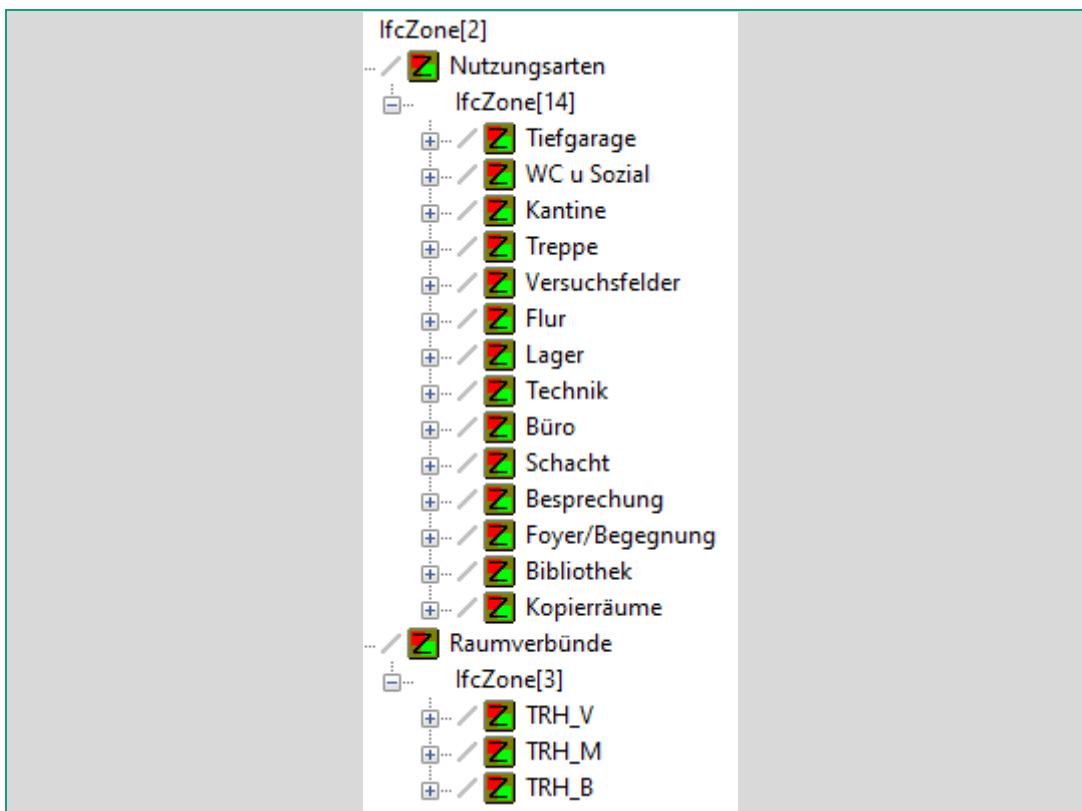


Abb. 18 Strukturierung der Räume eines Modells nach unterschiedlichen Kriterien (Screenshot aus KIT-Viewer)

2.1.9.3 Gruppierung von Elementen

Neben der Einordnung in die räumliche Struktur des Gebäudes (wie in Abschnitt 2.1.9 beschrieben) und der Zuordnung von Objekten zu Systemen (wie in Abschnitt 2.1.9 beschrieben) ist es gemäß IFC-Schema auch möglich, **IfcObjects** zu allgemeinen **IfcGroups** zuzuordnen. **IfcObjects** können einer, mehreren oder keiner Gruppe zugeordnet sein. Derartige Gruppierungen sind nicht hierarchisch. Gruppen können ineinander verschachtelt sein. Abb. 19 zeigt die dafür vorgesehenen Klassen im IFC-Standard.

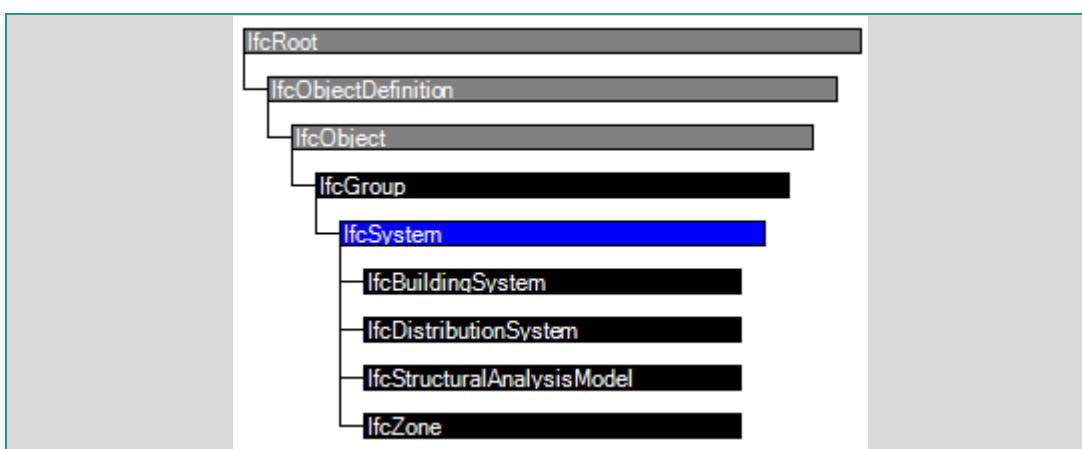


Abb. 19 Subklassen und Superklasse von **IfcSystem**

1.1.1.2 Repräsentation von Heizkreisen im IFC

Auf Seite 28 wurde das Konzept des Heizkreises eingeführt und seine Rolle als Schnittstelle zwischen Netz- und Zentralmodell hervorgehoben. Für die in dieser Arbeit thematisierten Fälle der

Anlagensimulation (Szenario „Variantenstudie Anlagentechnik“) ist die Repräsentation aller Komponenten eines Heizkreises als Platzhalter oftmals ebenso sinnvoll wie bei der Darstellung im Schaltschema. Eine dafür geeignete Entität im IFC-Schema muss einen IfcDistributionPort enthalten können, sowie eine Repräsentation haben können, daher sind alle Subklassen von IfcProduct (Abb. 14) denkbar.

Aufgrund der Bezeichnung kommen verschiedene IFC-Entitäten in Frage, die in Tab. 19 gegenübergestellt werden.

Tab. 19 Möglichkeiten zur Repräsentation von Heizkreisen in IFC

	predefinedTypes (außer USERDEFINED, NOTDEFINED)	Definition
<i>IfcVirtualElement</i>	keine	„A virtual element is a special element used to provide imaginary boundaries, such as between two adjacent, but not separated, spaces. Virtual elements are usually not displayed and does not have quantities and other measures.“
<i>IfcUnitaryEquipment</i>	AIRHANDLER AIRCONDITIONINGUNIT DEHUMIDIFIER SPLITSYSTEM ROOFTOPUNIT	„Unitary equipment typically combine a number of components into a single product, such as air handlers, pre-packaged rooftop air-conditioning units, heat pumps, and split systems.“
<i>IfcHeatExchanger</i>	PLATE SHELLANDTUBE	„A heat exchanger is a device used to provide heat transfer between non-mixing media such as plate and shell and tube heat exchangers.“
<i>IfcBuildingElementProxy</i>		“The IfcBuildingElementProxy is a proxy definition that provides the same functionality as subtypes of IfcBuildingElement, but without having a predefined meaning of the special type of building element, it represents.“

Das *IfcVirtualElement* ist für die Bereitstellung imaginärer Begrenzungen vorgesehen. Die praktische Anwendung erfolgt derzeit vorrangig für sog. „Raumbegrenzungslinien“ - die Definition lässt jedoch auch weitere Verwendungen zu. Problematisch für die Handhabung ist, dass die meisten IFC-Viewer diese Entität nicht darstellen (siehe Tab. 6).

Alternativ wäre eine Abbildung als Entität in der *IfcDistributionElement*-Hierarchie (siehe Abb. 16) denkbar. *IfcUnitaryEquipment* und *IfcHeatExchanger* sind dabei naheliegend. Die Definition und die zugehörige PredefinedType-Enumeration zeigen, dass beide Entitäten eigentlich für eine andere Verwendung vorgesehen sind. Die Entität *IfcBuildingElementProxy* hingegen ist kein *IfcDistributionElement*, sondern ein *IfcBuildingElement* und würde damit zu einer irreführenden Einordnung der Heizkreise in die Klassenhierarchie führen. Ein Pendant („*IfcDistributionElementProxy*“) in der *IfcDistributionElement*-Hierarchie existiert weder in IFC Version 4.0.2.1, noch in IFC 4.3.

Die Verwendung der Klassen *IfcDistributionElement*, *IfcDistributionFlowElement*, *IfcEnergyConversionDevice* wäre möglich, da diese Klassen nicht als *abstract* definiert sind. Eine Zuordnung von PredefinedTypes ist dazu jedoch nicht möglich, da entsprechende Attribute in der Klassenhierarchie erst für die in Tab. 19 genannten Entitäten vorgesehen sind. Für die Übersetzungen ist eine reine Zuordnung über Klassen i.d.R. aber nicht ausreichend (siehe Abschnitt 3.8.4.2), weshalb eine Entität gewählt wird, der *predefinedTypes* zugeordnet werden können.

In dieser Arbeit wird die Klasse *IfcHeatExchanger* mit dem *predefinedType* *USERDEFINED* für die Abbildung von Heizkreisen genutzt.

2.1.10 Geometriedarstellung in IFC

2.1.10.1 Darstellungskontexte

Jede Darstellung eines *IfcProduct* wird einem *IfcGeometricRepresentationContext* zugeordnet, dabei wird unterschieden nach zweidimensionalen (*ContextType=Plan*) und dreidimensionalen (*ContextType=Model*). Ein *IfcProject* kann beliebig viele *IfcGeometricRepresentationContext* enthalten. Neben der Dimension ist ein *IfcGeometricRepresentationContext* durch sein Koordinatensystem (Ursprung und Ausrichtung) und seinen Auflösung (*Precision*) gekennzeichnet, diese bezeichnet die Toleranz innerhalb derer zwei Punkte als einer interpretiert werden. Es sollte eine aussagekräftige Bezeichnung als *ContextIdentifier* vergeben werden.

Für jeden *IfcGeometricRepresentationContext* können beliebig viele *IfcGeometricRepresentationSubContexts* definiert werden. Diese zeichnen sich durch einen *TargetScale* und einen *TargetView* aus und sind daher in zweidimensionalen *IfcGeometricRepresentationContexten* relevant, z.B. um Grundrisse und Ansichten ggf. in unterschiedlichen Maßstäben zu unterscheiden. Wesentliche vordefinierte *TargetViews* sind in der *IfcGeometricProjectionEnum* enthalten: *PLAN_VIEW* (Grundriss), *SECTION_VIEW* (Schnitt), *ELEVATION_VIEW* (Ansicht). Für dreidimensionale Darstellungen ist *MODEL_VIEW* vorgesehen. Für vereinfachende/abstrakte/schematische Darstellungen sind *GRAPH_VIEW* und *SKETCH_VIEW* vordefiniert.

IFC-Viewer sollten dem Nutzer ermöglichen die Darstellungskontexte zu wählen, dies gewährleistet derzeit nur der FZK- bzw. KIT-Viewer (siehe Tab. 6). Abb. 20 zeigt die Auswahl beispielhaft im KIT-Viewer.

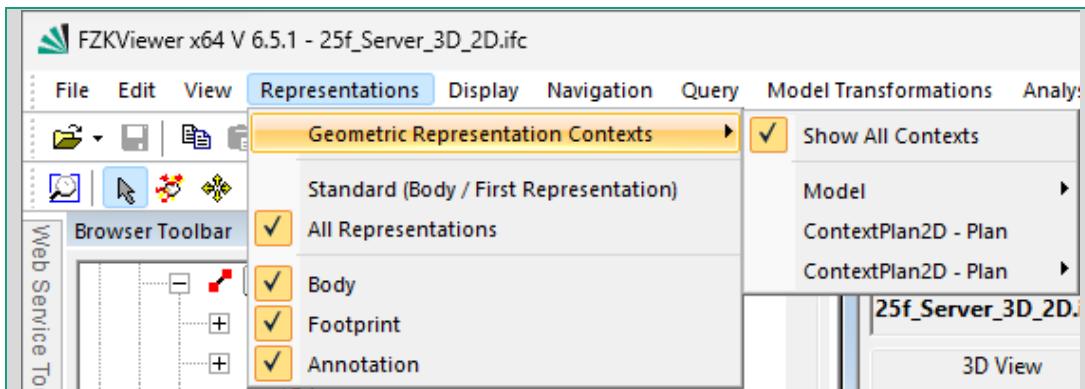


Abb. 20 Auswahl der Darstellungskontexte im FZK-Viewer

2.1.10.2 Repräsentationen

Darstellungen eines *IfcProduct* werden als *IfcRepresentation* bezeichnet. Diese haben einen *Identifier* und *Type* und bestehen aus einem oder mehreren *IfcRepresentationItems*. Es wird unterschieden in *IfcShapeRepresentation* und *IfcTopologyRepresentation*. *IfcShapeRepresentation* dienen dabei der Darstellung von jeglicher Art von Geometrie und werden daher im Folgenden ausschließlich weiter betrachtet. Die Liste der *IfcRepresentationItems* enthält die eigentliche Geometrie. Jede *IfcRepresentation* ist einem *IfcRepresentationContext* (s.o.) zuzuordnen. Für die Attribute *RepresentationIdentifier* und *RepresentationType* sollen möglichst die vordefinierten Werte verwendet werden. Sinnvolle Kombinationen im Kontext dieser Arbeit sind in Tab. 20 zusammengefasst. Ein Beispiel für unterschiedliche geometrische Repräsentation zeigt Abb. 21. Ein deutlicher Unterschied in der Dateigröße geht mit den unterschiedlichen Repräsentationen einher.

2 Methodische und Technische Grundlagen

2.1 IFC

2.1.10 Geometriedarstellung in IFC

Verschiedene Darstellungen eines *IfcProduct* werden von der zugeordneten *IfcProductDefinitionShape* zusammengefasst. Diese wird als Attribut dem *IfcProduct* zugewiesen.

Tab. 20 Zuordnung von *IfcRepresentationContext*, *RepresentationIdentifier*, *RepresentationType* und *IfcRepresentationItems* zu verschiedenen Darstellungsformen

Art der Darstellung	IfcRepresentation / IfcShapeRepresentation			
	Sinnvoller <i>IfcRepresentationContext</i> und ggf. <i>IfcRepresentationSubContext</i>	<i>RepresentationIdentifier</i>	<i>RepresentationType</i>	Sinnvolle Klassen für <i>IfcRepresentationItems</i>
3D-Geometrie in verschiedenen Detaillierungsgraden	<i>ContextType=Model</i>	für die echte/Haupt-Geometrie: <i>Body</i> für Freihaltevolumina: <i>Reference/Clearance</i>	<i>SweptSolid</i> , <i>Brep</i> , <i>CSG</i> , ...	<i>IfcSolidModel</i> (abstrakte Klasse für jede Art von 3D-Geometrie)
3D-Grobgeometrie mit Hilfe von <i>BoundingBox</i> /Umfassungskörpern	<i>ContextType=Model</i>	<i>Box</i>	<i>BoundingBox</i>	<i>IfcBoundingBox</i> (achsparalleler Quader zum Koordinatensystem)
2D Grundriss/Ansicht/Schnitt	<i>ContextType=Plan</i> <i>TargetView:</i> Grundriss: <i>PLAN_VIEW</i> Schnitt: <i>SECTION_VIEW</i> Ansicht: <i>ELEVATION_VIEW</i>	<i>FootPrint</i>	<i>Curve2D</i> , <i>Surface2D</i> , <i>FillArea</i> (Schraffur)	<i>IfcCurve</i>
		<i>Annotation</i>	<i>Text</i>	<i>IfcTextLiteral</i>
2D Grobgeometrie (Äquivalent zur <i>BoundingBox</i>)	<i>ContextType=Plan</i> Grundriss: <i>TargetView=PLAN_VIEW</i>			<i>IfcPlanarExtent</i> (achsparalleles Rechteck zum Koordinatensystem)
2D Schema	<i>ContextType=Plan</i> <i>TargetView:</i> <i>GRAPH_VIEW</i> oder <i>SKETCH_VIEW</i>	<i>FootPrint</i>	<i>Curve2D</i> , <i>Surface2D</i> , <i>FillArea</i> (Schraffur)	<i>IfcCurve</i>
		<i>Annotation</i>	<i>Text</i>	<i>IfcTextLiteral</i>

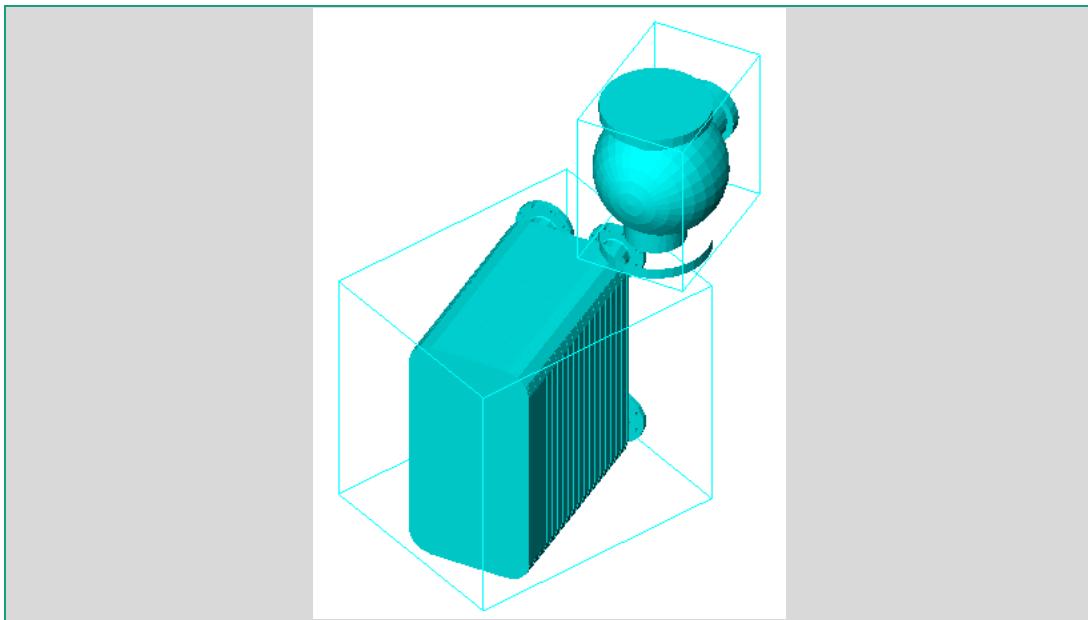


Abb. 21 Gegenüberstellung Originalgeometrie und BoundingBox

2.1.10.3 Platzierung

Die Platzierung eines *IfcProduct* wird i.d.R. mit einem Objekt der Klasse *IfcLocalPlacement* festgelegt. Diese ermöglicht es, ein Bezugskoordinatensystem anzugeben und die Platzierung relativ dazu vorzunehmen. Wird kein Bezugskoordinatensystem explizit angegeben, erfolgt die Platzierung in Bezug auf das Koordinatensystem des zugeordneten *IfcGeometricRepresentationContext* bzw. des *IfcGeometricRepresentationSubContext*. Üblich ist eine Platzierung relativ zur beinhaltenden Struktur, z.B. ein Raum in einer Etage, ein Heizkörper in

einem Raum, jedoch sind theoretisch beliebige Verschachtelungen von Bezugskoordinatensystemen denkbar und standardkonform.

2.2 Modelica

2.2.1 Überblick

Modelica ist eine objektorientierte Modellierungssprache für multiphysikalische Systeme. Sie unterstützt die Verbindung von Komponenten, die durch mathematische Gleichungen abgebildet werden und erleichtert damit insb. das Erstellen von White-Box-Modellen. Objektorientierte Konstrukte erleichtern die Wiederverwendung von Modellen und die Erstellung komplexer Modelle, insbesondere auch solcher Modelle, die Teilkomponenten aus verschiedenen Domänen z.B. Mechanik, Thermodynamik und Elektrotechnik enthalten.

Die Verbindung der Komponenten erfolgt dabei aksausal: Eine Verbindung ist durch einen Satz Gleichungen definiert, welche erfüllt sein müssen. In der Regel handelt es sich dabei um eine Gleichsetzung von Potentialgrößen und eine Summierung der Flussgrößen. Dieses Modellierungskonzept unterscheidet sich damit fundamental von der Signalbasierten Modellierung (Input-Verarbeitung-Output) wie sie z.B. aus MATLAB bekannt ist.

Modelica wurde 1997 von der Modelica Association vorgestellt und wird seitdem von dieser gepflegt. Die Entwicklung erfolgt quelloffen und die Sprachspezifikation ist unter freier Lizenz verfügbar. Wesentliche Komponenten von Modelica sind

- die Modelica Language Specification MLS (derzeit aktuell Version 3.6 vom März 2023) und
- die Modelica Standard Library MSL (derzeit aktuelle Version 4.0.0 vom Juni 2020).

2.2.2 Sprachspezifikation

Die Sprachspezifikation beschreibt die Syntax des Modelica-Formats inkl. der Schlüsselwörter. Wesentliche Inhalte, welche für das Verständnis dieser Arbeit relevant sind, werden im Folgenden wiedergegeben.

2.2.2.1 Klassen

Die fundamentale Struktureinheit für die Modellierung in Modelica sind **Klassen**, dafür sind die synonymen Schlüsselwörter *model* und *class*²³ vorgesehen. In diesem Text wird für dieses Konzept ausschließlich der Begriff *Klasse* benutzt, um Verwechslungen mit dem ebenfalls sehr wichtigen Konzept *Modell* auszuschließen. Klassen können Gleichungen und algorithmischen Code enthalten, außerdem enthalten sie i.d.R. **Komponenten**.

Modelica Klassen liegen als ASCII-Dateien vor, die Standard-Datei-Endung lautet „*.mo“. Derartige Dateien werden in dieser Arbeit als **MO-Dateien** bezeichnet. Die Modellierung kann durch Manipulation des Quelltextes erfolgen. Dazu ist kein spezielles Modelica-Tool notwendig, dies kann mit beliebigen Texteditoren erfolgen.

Packages sind als Klasse dafür vorgesehen, Modelica-Klassen zu gruppieren und können ineinander verschachtelt werden. Etablierte Praxis ist, für jede Klasse eine eigene MO-Datei anzulegen, Packages bilden sich dann auf dem Dateisystem als Ordnerstrukturen ab. Die Ordner müssen eine *package.mo*-Datei enthalten, welche die Metainformationen zum Package (wie Documentation und Icon-Layer) enthält.

²³ Das Schlüsselwort *class* war nur bis MLS 3.4 zulässig, inzwischen wird ausschließlich *model* verwendet

Klassen sollen i.d.R. ausgeglichen (*balanced*) sein: Das bedeutet, dass die Anzahl der Variablen in dieser Klasse gleich der Anzahl der Gleichungen ist. Im Gegensatz dazu stehen partielle *partial* Klassen: Derartige Klassen können nicht instantiiert werden, sind jedoch hilfreich als Basisklassen einer Vererbungshierarchie²⁴.

In der Modelica Sprachspezifikation (MLS) sind verschiedene Klassen definiert. Die Schlüsselwörter *class* bzw. *model* bezeichnen dabei die allgemeinste Klasse ohne weitere Restriktionen. Speziellere Klassen, welche in dieser Arbeit Verwendung finden, und ihre Restriktion sind in Tab. 21 zusammengestellt.

Tab. 21 Restriktionen wichtiger Klassen in Modelica

Schlüsselwort	Beschreibung	equation	algorithms	Konnektoren	Sonstige Komponenten
<i>model</i>	Allgemeine Klasse	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<i>record</i>	Für die Bündelung von Werten	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<i>block</i>	Für Modellierung mittels Blockschaltbild	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Müssen Richtung input/output haben	
<i>function</i>	Ohne inneren Zustand	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Müssen Richtung input/output haben	<input checked="" type="checkbox"/>
<i>connector</i>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
<i>package</i>	Darf nur Klassendefinitionen enthalten	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Klassen sind charakterisiert durch ihren Parameter und Konnektoren. Eine Kombination aus Parameters und Konnektoren wird als *shape* einer Klasse bezeichnet. Komponenten mit derselben shape sind gegeneinander austauschbar.

Eine spezielle Art von Modellen sind **ausführbare Simulationsmodelle**. Diese zeichnen sich durch eine ausgeglichene Anzahl von Unbekannten und Gleichungen („balanced model“), sowie das Fehlen von Eingängen aus. In der MLS werden diese als „simulation model“ bezeichnet - da dies in dieser Arbeit Verwechslungsgefahr mit dem Allgemeinen Begriff „Simulationsmodell“ birgt, werden sie in dieser Arbeit als **Modelica-Executables** bezeichnet. Die Konvention in der MSL ist, derartige Modelle mit einem „Play“-Icon zu versehen.

2.2.2.2 Komponenten

Um **Komponenten** in Klassen zu erzeugen müssen diese deklariert werden, d.h. Bezeichnung *identifier* und eine Klassenzuordnung *type* müssen festgelegt werden. Eine Komponente kann – muss aber nicht – initialisiert werden.²⁵ Komponenten können Skalare oder Arrays sein. Komponenten können wie folgt gruppiert werden:

- Primitive Komponenten sind Instanzen von primitiven Klassen
 - Parameter (unveränderlich während der Laufzeit), Schlüsselwort: *parameter*
 - Variablen (zeitlich veränderlich)
- Komplexe Komponenten sind Instanzen von komplexen Klassen. Komplexe Klassen bündeln wiederum Parameter, Konnektoren, Variablen und weitere Komponenten. Ein wichtiger Sonderfall der komplexen Komponenten sind die sog. Konnektor-Komponenten. Diese dienen der Verbindung von komplexen Komponenten untereinander

2.2.2.3 Konnektoren

Konnektoren bilden die Schnittstelle einer Klasse nach außen. Sie sind als komplexe Komponente in einem Modell enthalten. Abgesehen von den weit verbreiteten *RealInput*, *RealOutput*, *BooleanInput*, *BooleanOutput* sind Konnektoren i.d.R. aksausal, d.h. es ist keine Richtung des

²⁴ In anderen Programmiersprachen wird ein derartiges Konstrukt als „abstrakte Klasse“ bezeichnet.

²⁵ Modelica verhält sich hier analog zu Java und anders als Python. („typsicher“)

Informationsflusses vorgegeben. Konnektoren sind eine spezielle Klasse in Modelica, welche mit dem Schlüsselwort `connector` gekennzeichnet sind. Sie enthalten keine Gleichungen oder Algorithmen, sondern ausschließlich Variablen. Diese werden unterteilt in Potential-, Fluss- und Streamvariablen, letztere werden mit den Schlüsselwörtern `flow` und `stream` markiert. Flussvariablen haben die Eigenschaft, dass sie sich an Knotenpunkten zu null addieren, während Potentialvariablen an Knotenpunkten für alle angrenzenden Komponenten gleich sind (Analogie zum Knoten- und Maschensatz in der Elektrotechnik und Hydraulik). Für die Fluid-Konnektoren, die in dieser Arbeit i.d.R. verwendet werden, sind außerdem Streamvariablen wichtig. Diese beschreiben Größen, welche von einer Flussvariable getragen werden, jedoch abhängig sind von der Fließrichtung im Knotenpunkt, da sie vom „Oberlauf“ definiert werden, z.B. Partikelkonzentrationen oder der Wasserdampfanteil in feuchter Luft. Tab. 22 stellt wichtige Konnektoren gegenüber.

Tab. 22 wichtige Modelica Konnektoren

	FluidPort	HeatPort	Input/Output
Icon			
Pfad	Modelica.Fluid.Interfaces	Modelica.Thermal.HeatTransfer.Interfaces	Modelica.Blocks.Interfaces
Potentialgröße	Druck	Temperatur	- (da kein physikalischer, sondern ein kausaler Konnektor)
Flussgröße	Massestrom	Wärmestrom	- (da kein physikalischer, sondern ein kausaler Konnektor)

2.2.2.4 Zuordnung von Eigenschaften zu Modelica-Objekten

Modelica-Objekten können auf verschiedene Art und Weise Eigenschaften zugeordnet werden. Klassen haben **Parameter**, welche vom Modelica-Translator (siehe Abschnitt 2.2.5) in das zu lösende Gleichungssystem eingebaut werden. Die Modelica-Anwendungen behandeln auch sog. **Dokumentations-Strings** `stringComments` und **Annotationen** `annotations`, welche jeweils für Komponenten und Klassen angegeben werden können. Die MLS sieht dabei verschiedene Arten von Annotationen vor. Wichtig sind insb. die Annotationen zur Zuordnung der alternativen Model-Layer (*Icon*, *Diagram*, *Documentation*), die in Abschnitt 2.2.4 beschrieben werden. Weiterhin existieren vordefinierte Annotation zur Definition des Simulationsexperiments bei Executables (Start- und Endzeit, Zeitschrittweiten, Präzision etc.), zum Versionshandling und zur Steuerung von Zugriffsrechten. Es ist außerdem möglich, herstellerspezifisch weitere Annotationen zu ergänzen, diese sind dann allerdings nicht zwischen verschiedenen Simulatoren kompatibel. Mittels Annotationen für Auswahllisten bei Modifikation und Redeklarationen, sowie für die GUI, sind auch die zu einer Klasse gehörigen Parameterdialoge herstellerneutral in der Modelica-Datei abspeicherbar.

Mittels **Quelltext-Kommentaren** können Objekten zusätzlich Eigenschaften zugeordnet werden. Diese werden weder vom Modelica-Translator noch von der Modelica-Anwendung interpretiert.

2.2.3 Simulationswerkzeuge

Es existieren verschiedene Simulationswerkzeuge, um mit Modelica-Modellen zu arbeiten. Bei allen bereits in Tab. 7 genannten Modelica-basierten Simulatoren handelt es sich um integrierte Simulatoren, d.h. die drei notwendigen Bestandteile (siehe Abschnitt 1.3.2) Simulationskern, Benutzeroberfläche und Simulationsbibliotheken sind enthalten. Die Simulatoren unterstützen den Nutzer bei allen notwendigen Schritten bei der Arbeit mit Modelica-Modellen wie in Abb. 22 dargestellt.

2 Methodische und Technische Grundlagen

2.2 Modelica

2.2.3 Simulationswerkzeuge

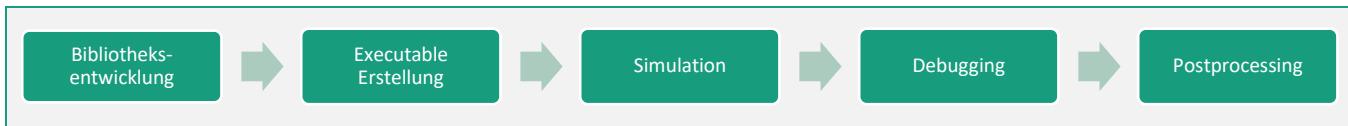


Abb. 22 Funktionsumfang von Modelica-Tools

Die Simulatoren unterscheiden sich im Wesentlichen hinsichtlich der Kosten, der Kompatibilität mit Bibliotheksversionen und der enthaltenen **Solver**. Kommerzielle Tools haben i.d.R. leistungsstärkere Solver. Im Rahmen dieser Arbeit wurde Dymola in verschiedenen Versionen (2020-2023) verwendet. Im Vergleich dazu zeigt Tab. 23 den Funktionsumfang vom kostenlos und frei verfügbaren Tools OpenModelica. Dieses hatte in der Vergangenheit Kompatibilitätsprobleme mit den bei der Gebäude- und Anlagensimulation üblicherweise verwendeten Bibliotheken (siehe Abschnitt 2.2.6), in den aktuellen Versionen (siehe Tab. 24) jedoch nicht mehr.

Tab. 23 Funktionsumfang von Modelica-Tools im Vergleich

	Dymola	OpenModelica
Hersteller	Dassault Systems	Open Source Modelica Consortium OSMC
Aktuelle Version	Dymola 2023 (13.5.22)	1.21.0 vom 18.4.23
Kompatibilität	MSL4.0.0, MBL 9.0.0	MSL4.0.0, MBL 9.0.0
Verfügbarkeit	Via Reseller vor Ort	https://openmodelica.org/
Unterstützte Betriebssysteme	Windows, Linux	Windows, Linux, Max, Docker
Kosten	>>1000€/a	kostenlos
Funktionsumfang	Bibliotheksentwicklung	integriert
	Executable Erstellung	Integriert
	Simulation	Integriert
	Debugging	Integriert
	Postprocessing	Integriert
	Interne Shell	vorhanden
	FMU-Export*	Integriert (ggf. extra Lizenz nötig)
	Sonstiges	OMNotebook, OMoptim
	Nutzbarkeit von der CLI des Betriebssystems	<input checked="" type="checkbox"/>
Enthaltene Solver	Dassl	DASSL
	Cvode	CVODE
	Euler	EULER
	Lsodar	Impeuler
	Rkfix2, 3, 4	IDA
	Radau	GBODE
	Esdirk23a, 34a, 45a	HEUN
	Dopri45, 853	Rungekutta, Imprungekutto, rungekuttaSSc
	Sdirk34hw	Trapezoid
	Cerk23, 34, 45	irkscos
		symSolver, symSolverSSc
		qss

*Functional Mockup Unit (FMU) ist ein Standard für Komponenten von gekoppelten Simulationen

Abb. 23 zeigt die Benutzeroberfläche von Dymola 2022. Anhand dessen sollen typische Funktionen von Modelica-Simulatoren erläutert werden. Die Benutzeroberfläche von OpenModelica hat einen ähnlichen Aufbau. In der Regel haben die Tools umschaltbare Arbeitsbereiche für die Modellerstellung und die anschließende Simulation mit Postprocessing. Im mittleren Bereich wird das gerade aktuelle Modell angezeigt. In diesem Fall ist der *Diagram Layer* aktiv. Im oberen Bereich befinden sich Umschaltmöglichkeiten zwischen den verschiedenen Layern (*Graphics: Icon + Diagram, Documentation, Text*).

Im linken Bereich befindet sich der sog. Package Browser bzw. die Liste der geladenen Libraries. Von dieser aus können Komponenten per Drag&Drop im Modell erzeugt werden. Verbindungen zwischen Komponenten im Diagram werden einfach durch Anklicken beider zu verbindender

Konnektoren erzeugt. Im rechten Bereich befindet sich die sog. Komponentenliste, welche alle im Modell vorhandenen Komponenten aufführt, auch jene, die keine grafische Repräsentation haben.

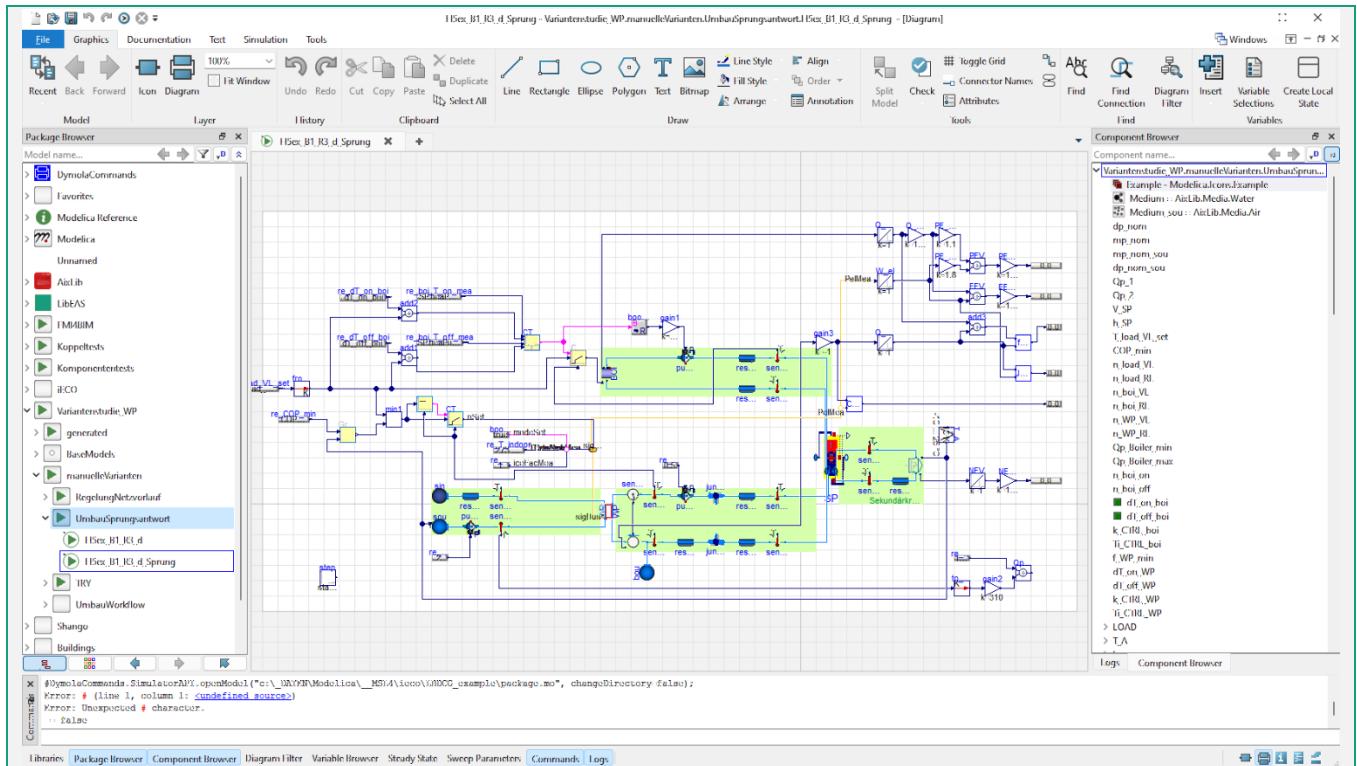


Abb. 23 Benutzeroberfläche des Tools Dymola 2022 mit dem Arbeitsbereich für die Modellerstellung

2.2.4 Model-Layer

Modelica-Simulatoren erlauben alternativ zur textbasierten Manipulation der MO-Dateien die grafische Bearbeitung der Modelle in einer Art Block-Schaltbild - Abb. 24 zeigt diesen sog. „Diagram Layer“ (im Folgenden kurz als „**Diagram**“ bezeichnet) eines Beispielmodells. Diese Ansicht enthält die Icons der im Modell enthaltenen Komponenten, sowie Verbindungslien für verbundene Konnektoren. Dabei handelt es sich in der Regel nicht um ein Blockschaltbild im Sinne der Regelungstechnik, da die Verbindungen i.d.R. ungerichtet d.h. aksual sind (hellblaue Linien in Abb. 24). Eine Ausnahme bilden die Real-In- und -outputs, welche in Abb. 24 als dunkelblaue Linien dargestellt sind. Diese sollten im Falle einer White-Box-Modellierung nur für Regelungsein- und -ausgänge zur Anwendung kommen, da nur diese auch in der Realität gerichtete Signalverbindungen sind. Im Gegensatz dazu sind beispielsweise hydraulische Verbindungen (Rohre) ungerichtet, da sie in beiden Richtungen durchströmt werden können.

Die Informationen zur Darstellung im Diagram werden im Modell selbst als sog. **Annotationen** gespeichert. Diese sind ebenfalls in der MLS definiert, weshalb auch die grafische Darstellung eines Modells kompatibel zwischen verschiedenen Modelica-Tools ist. Ein weiterer Vorteil im Kontext dieser Arbeit ist, dass alle Informationen zu einem konkreten Modell in der Modeldatei enthalten sind.

Neben dem **Diagram** und dem **Text-Layer** existieren zwei weitere Layer für **Icon** und **Documentation**, auch diese werden über Annotationen realisiert und liegen somit in dem Modelldateien selbst vor. Annotationen sind wie alle anderen Modelleigenschaften auch Bestandteil der Vererbungshierarchie. Dementsprechend können auch (Teile von) Icons mit geerbt werden.

Modelica-Tools unterstützen den Nutzer bei der Erzeugung der verschiedenen **Layer**:

- Automatische Erstellung von Dokumentationsrumpfen anhand der Package-Struktur, enthaltener Parameter und Konnektoren, Vererbungsstruktur und sonstiger Metainformationen.
- Automatisches rechtwinkliges Routing der Verbindungslinien im Diagram
- Farbliche Codierung der Verbindungslinien entsprechend der Art der verbundenen Konnektoren
- Automatische Beschriftung der Komponenten im Diagram
- Automatische Generierung des Icon-Layers anhand enthaltener Konnektoren

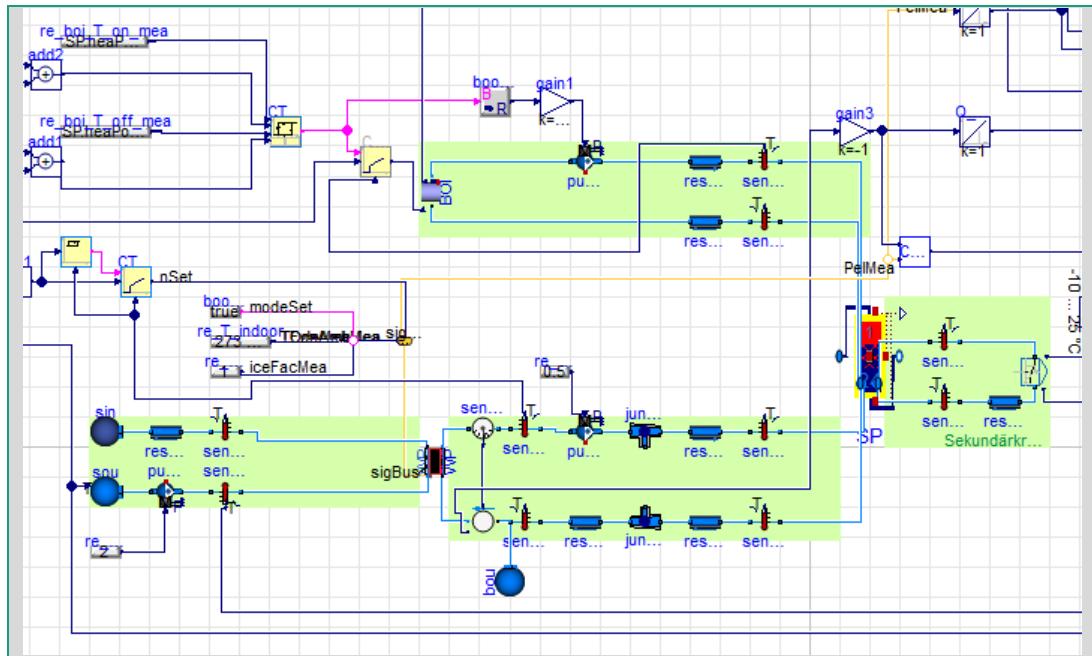


Abb. 24 Diagram Layer eines Modelica-Modells (Ausschnitt)

Das problemlose Wechseln zwischen grafischer und textbasierter Modellbearbeitung ist einer der wesentlichen Vorteile der Arbeit mit Modelica. Es ermöglicht das Arbeiten auf der jeweils benötigten Detaillierungsebene. Die grafische Ebene ist geeignet, um schnell einen Überblick über ein vorhandenes Modell zu gewinnen. Detaillierte Meta-Informationen können im Documentation-Layer hinterlegt werden. Reicht beides nicht aus, kann der Nutzer jederzeit auf den **Text-Layer** wechseln und das Modell in beliebiger Detailtiefe untersuchen und auch anpassen.

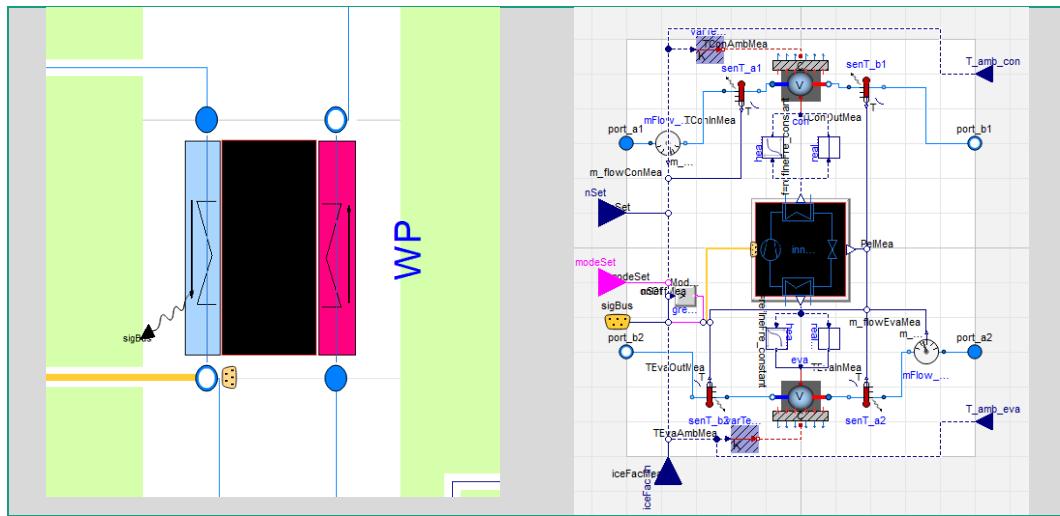


Abb. 25 Icon und Diagramm Layer eines Beispielmodells (AixLib.Fluid.HeatPumps.HeatPump)

Das Konzept zweier grafischer Layer (Icon und Diagram, beispielhaft in Abb. 25 dargestellt) ist eine ideale Unterstützung zur grafischen Bearbeitung stark gekapselter Modelle. Simulatoren ermöglichen es i.d.R. zu Modellkomponenten, die durch ein Icon repräsentiert werden, an anderer Stelle das Diagramm anzuzeigen, damit wird ein hineinzoomen in die Modelle ermöglicht.

2.2.5 Ablauf der Simulation

Abb. 26 zeigt die Prozessschritte zur Simulation eines Modelica-Modells. Zunächst werden alle Referenzen aufgelöst, d.h. die Gleichungen, welche in den Klassen der enthaltenen Komponenten definiert wurden, werden im *Executable* zusammengetragen und entsprechend parametriert bzw. die Eingänge und Ausgänge gesetzt. Dabei wird symbolisch umgeformt, es erfolgen keine numerischen Operationen. Dieser Vorgang wird als **Flattening** oder **Translation** bezeichnet. Dadurch entsteht ein hybrides Differential-Algebraisches Gleichungssystem (DAE).

Im nächsten Schritt folgt die **Initialisierung**, d.h. es wird ein konsistenter Satz an Startwerten für alle im Modell vorhandenen nichtlinearen Variablen gesucht. Dabei werden ggf. vom Nutzer vorgegebene Startwerte berücksichtigt. Dieser Schritt kann sehr aufwändig sein und auch scheitern. Hilfreich sind konsistente und sinnvolle Nutzervorgaben für Startwerte.

Anschließend erfolgt die eigentliche **Simulation**, indem das aufgestellte Gleichungssystem für den vorgegebenen Zeitraum mit Hilfe numerischer Integration gelöst wird. Die Lösung des Gleichungssystems übernimmt der **Solver**. I.d.R. kommen Solver mit variabler Zeitschrittweite zum Einsatz: Der Solver wählt die Zeitschrittweite nur so klein, wie es zum Lösen des Systems zu einem bestimmten Zeitpunkt mit der vom Nutzer vorgegeben Toleranz notwendig ist. Ggf. erfolgt eine iterative Verkleinerung der Zeitschrittweite. Die Komponente mit der kleinsten Zeitkonstante bestimmt dabei die vom Solver zu wählende Zeitschrittweite – häufig sind dies in der Anlagensimulation die Regelungskomponenten mit deutlich geringeren Zeitkonstanten als die geregelten thermisch tragen Anlagen. Alternativ können auch Solver mit fester Zeitschrittweite verwendet werden.



Abb. 26 Prozessschritte und Zwischenergebnisse bei der Bearbeitung eines Executable Modelica-Modells

Als Ergebnis entstehen die Lösungstrajektorien d.h. für jede beteiligte Variable wird der zeitliche Verlauf ermittelt. Ohne **Postprocessing** sind die Trajektorien bei Solvern mit variabler Zeitschrittweite nicht äquidistant. Je nach verwendetem Simulator sind die Ergebnisse der Zwischenschritte gemäß Abb. 26 ggf. als Dateien auf dem Dateisystem verfügbar.

2.2.6 Bibliotheken

Ein wesentlicher Vorteil von Modelica ist die Verfügbarkeit hochentwickelter Bibliotheken. Es kann unterschieden werden zwischen kommerziellen Bibliotheken, deren Quellcode i.d.R. nicht offen zur Verfügung steht, und FLOSS-Bibliotheken. Im Rahmen dieser Arbeit kamen ausschließlich letztere zum Einsatz.

Der Begriff „Bibliothek“ ist in der MLS nicht definiert. In dieser Arbeit wird darunter eine Sammlung von Modelica-Klassen verstanden, welche über eine spezielle Aufgabe hinaus Anwendung finden soll und dementsprechend zur Verfügung gestellt werden. Eine Sammlung von Klassen wird in Modelica mit „Packages“ realisiert.

2.2.6.1 MSL

Unabhängig von der Simulationsdomäne kommt die **Modelica Standard Library** zur Anwendung. Abb. 27 zeigt einen Überblick der enthaltenen Subpackages. In dieser Arbeit sind Komponenten aus dem *Modelica.Blocks*-Package wichtig für die Abbildung einfacher Randbedingungen und der Regelungstechnik.

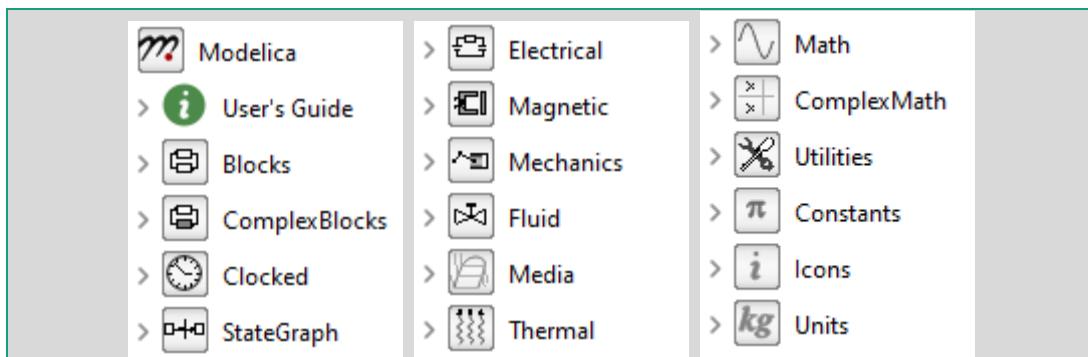


Abb. 27 Packages in der MSL

2.2.6.2 IBPSA

Für Gebäude- und Anlagensimulation existieren spezielle **Domänenbibliotheken**. Von besonderer Bedeutung sind dabei die sog. **IBPSA-Libraries**. Diese entstanden in den Jahren 2012-2017 im Rahmen des Annex60-Projekts (Wetter und van Treeck 2017) des „Energy in Buildings and Communities Programme“ der Internationalen Energieagentur (IEA EBC) durch die Harmonisierung von vier auch vorher schon vorhandenen FLOSS-Gebäude-Simulationsbibliotheken:

- AixLib, der RWTH Aachen: (AixLib)
- Buildings, des LBNL, Berkeley, CA, USA: (MBL)
- BuildingSystems, der UdK Berlin: (BS)
- IDEAS der KU Leuven, Belgium: (IDEAS)

Es wurde ein gemeinsamer Kern definiert, den fortan alle Bibliotheken teilen, dieser wird als „IBPSACore“ bezeichnet, da er im Rahmen des „Project 1“ („IBPSA Project 1“ 2021) von der International Building Performance Simulation Association (IBPSA) 2017 bis 2022 weiterentwickelt wurde. Außerdem wurde die Package-Struktur vereinheitlicht. Man erkennt dies in der Gegenüberstellung der Bibliotheken in Abb. 28. Die o.g. vier Bibliotheken werden in dieser Arbeit

als IBPSA-Libraries bezeichnet. Jede der Bibliotheken erweitert den gemeinsamen Kern um eigenentwickelte Simulationsmodelle, welche in Tab. 24 gegenübergestellt werden.



Abb. 28 Package Struktur der IBPSA Libraries
gleichnamige Packages sind farbig hervorgehoben und entstammen dem IBPSA core

Alle Bibliotheken teilen das Paradigma, dass stoffagnostisch modelliert wird und das konkret vorhandene Medium von Anwender als Parameter eingestellt werden kann. Stoffmodelle für die üblichen Anwendungen Luft, Wasser, Wasser-Glycol-Gemische sind in verschiedenen Detaillierungsstufen, sowohl in den IBPSA Libraries (Package *IBPSA.Media*), als auch in der MSL (package *Modelica.Media*), vorhanden.

Eine wesentliche gemeinsame Modellierungsgrundlage ist weiterhin die Nutzung der FluidPorts und HeatPorts aus der MSL, somit sind die Modelle aus verschiedenen Bibliotheken untereinander kompatibel. Tab. 24 zeigt, dass teilweise eigene Konnektoren definiert wurden. Damit ausgestattete Modelle sind außerhalb der Library inkompatibel. Dies betrifft jedoch nur den geringeren Teil der Modelle (z.B. elektrische Komponenten der MBL, Solarkomponenten der BS). Man erkennt, dass die *Buildings* und *AixLib* den größten Bestand an kompatiblen Modellen bereit stellen und am aktivsten gepflegt werden. Sie wurden daher für die weitere Verwendung in dieser Arbeit ausgewählt.

Detailliertere Ausführungen zu diesen Simulationsbibliotheken, sowie eine Bewertung in Hinblick auf die verschiedenen Simulationsszenarien finden sich in (Eckstädt, Huang, u. a. 2023).

2 Methodische und Technische Grundlagen

2.3 Semantische Technologien

2.3.1 Begriffe

Tab. 24 Gegenüberstellung der IBPSA-Bibliotheken

	Buildings	AixLib	BuildingSystems	IDEAS
Kürzel	MBL	aix	BS	IDEAS
github Repository	lbl-srg/ modelica-buildings	RWTH-EBC/ AixLib	Udk-VPT/ BuildingSystems	open-ideas/ IDEAS
Aktuelleste Version	10.0.0 5.9.2023	1.3.2 9.2.23	2.0.0 22.5.17	3.0.0 3.5.2022
Kompatibilität mit OpenModelica (nach Angabe der Autoren)	ja	0.86	Keine Angabe	Ja
Aktivität der Community*	Anzahl Commits 341	114	7	97
	Aktive Contributors 3	11	1	2
Inhalte	Gebäudeseite	Spawn of EnergyPlus	Quartiere HighOrder	Quartiere Gebäudetypen Deutschland
	Wärmeübergabe	Deckenstrahlplatten Bauteilintegrierte Heizung Luftheizer	Bauteilintegrierte Heizung Lüftungsgeräte	Deckenstrahlplatten Bauteilintegrierte Heizung
	Thermische Erzeuger	Absorptionskälteaschinen Kühltürme KWK Wärmeübertrager Wärmepumpen Erdsonden	KWK Wärmepumpen Kennfeldmodell Solarthermie	Solarthermie Kältemaschinen KWK Fernwärmestationen
	Elektrisch	Umfangreicher Katalog (1 und 3-phasisig) mit Erzeugern (PV, Windrad) und Umrichtern	PV-Paneele Wechselrichter	PV Batteriespeicher
	Hydraulik		Pumpenmodelle Hydraulische Gruppen	Erdverlegte Rohre
	Regelung	OpenBuildingControl	Komplexe Anlagenregelung	
	Sonstiges	Validierung ASHRAE36 HeatTransfer Umfangreiche Nutzungszeitpläne	Testmodelle ASHRAE140 Stoffmodelle für Kältemittel	Wetterdaten DWD, TRY HeatAirMoistureTransport
	Komplexbeispiele	Rechenzentrum	Einfamilienhaus	Nahwärmenetz Berlin
	Eigene Konnektoren	Nur für die elektrische Bibliothek	Keine mehr (entsprechende Bibliotheks- Komponenten sind inzwischen deprecated)	RadiationPort (und zahlreiche weitere)
				Eigene BusConnectoren

* (02/2023-01/2024)

**inzwischen wieder deprecated

2.3 Semantische Technologien

2.3.1 Begriffe

In den Abschnitten 1.2.3.3 und 1.4.3 wurden bereits Ontologien thematisiert. Diese gehören zu sog. Semantischen Technologien, da sie sich mit der maschinenlesbaren Abbildung von Bedeutung und Wissen beschäftigen. Die dafür notwendigen Grundlagen werden in den folgenden Abschnitten beschrieben.

Die Wissenspyramide (Abb. 29) zeigt die Zusammenhänge zwischen den Begriffen Daten, Informationen und Wissen. Man erkennt, dass **Informationen** aus **Daten** entstehen, indem **Bedeutung** zugewiesen wird. Informationen werden zu **Wissen**, indem sie mit anderen Informationen verknüpft werden. Die exakten Abgrenzungen dieser Begriffe untereinander sind Gegenstand philosophischer Diskussionen und im Kontext dieser Arbeit weniger relevant. Es ist jedoch wichtig festzuhalten, dass Wissen durch Verknüpfung von Informationen entsteht und dass Informationen aus Daten erst entstehen, wenn diesen Bedeutung zugeordnet wird. Die Zuordnung

von Kontext zu Daten erfolgt oft auf schematischer Ebene, etwa indem Daten in einer bestimmten Tabellenspalte enthalten sind.

Wissen besteht i.d.R. aus einer Verknüpfung von konkretem **Instanzwissen** und allgemeinem abstraktem **Schemawissen**. Instanzwissen ist z.B. das Wissen über Nennweite und Material bestimmter Rohrstücke. Schemawissen ist z.B. die Tatsache, dass Rohre Bestandteile von Heizkreisen sind und dass sie mit einem bestimmten Medium gefüllt sind.

Wissen kann auf Computern in verschiedenen Formen dargestellt werden, z.B. tabellarisch, als Baumstruktur und als Wissensgraph.

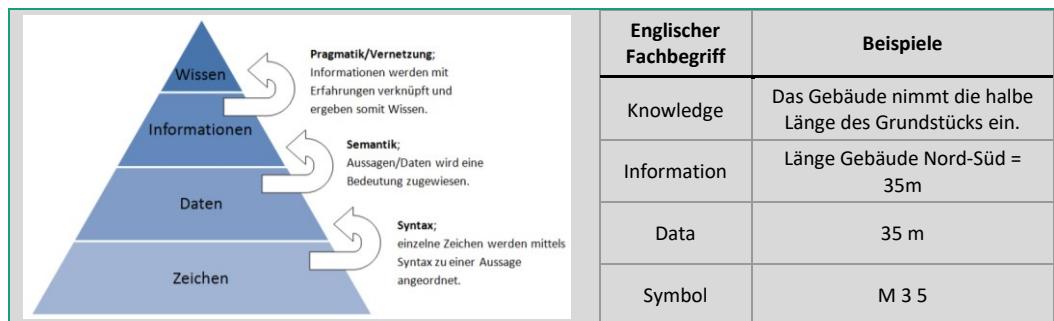


Abb. 29 Wissenspyramide, ergänzt nach (Herrmann 2012)

2.3.2 Wissensgraphen

2.3.2.1 Erläuterung des Konzepts

Graphen sind in der Informatik Datenstrukturen welche aus Knoten und Kanten bestehen. **Wissensgraphen** nutzen derartige Datenstrukturen um Wissen darzustellen. Das „Resource Description Framework“ (**RDF**) (siehe Tab. 54) ist ein W3C-Standard für die Beschreibung von Wissensgraphen. Demnach bilden sog. **Triple** (3-Tupel) die kleinste Informationseinheit. Ein Tripel besteht aus Subjekt, Prädikat und Objekt und wird genutzt um Aussagen über sog. **Ressourcen** zu treffen z.B.

Gebäude1 befindetSichIn Dresden.

Lis. 5 Triple als Pseudocode

Die beschriebenen Ressourcen können dabei aus dem Bereich des Instanz- oder Schemawissens stammen. Lis. 6 zeigt fünf Tripel. Die ersten beiden beschreiben Instanzwissen, zwei weitere beschreiben Schemawissen. Das letzte Triple stellt die Verbindung zwischen Instanz- und Schemawissen her. *Rohr_474* und *Rohr* sind die in diesem Beispiel beschrieben Ressourcen.

```

Rohr_474 hatMaterial PE-Xa
Rohr_474 hatNennweite DN12
Rohr istBestandteilVon Heizkreis
Rohr istGefüllt mit Medium
Rohr_474 istEin Rohr

```

Lis. 6 Beispiele für Instanz- (orange) und Schemawissen (blau) als Pseudocode

Das pinke Tripel zeigt eine Verbindung zwischen Instanz- und Schemawissen.

Das Subjekt repräsentiert dabei die in einem Triple beschriebene Ressource. Dieser wird mittels des **Prädikats** ein **Objekt** zugeordnet. **Objekte** können in RDF entweder sog. **Literale** (im Beispiel: *PE-Xa*, *DN12*) oder ebenfalls **Ressourcen** (*Rohr* im pinken Beispieltripel) sein. Literale können zusätzlich zum Datum Informationen zu Datentyp und ggf. Sprache enthalten, sie werden im Folgenden als **Data Type Properties** bezeichnet. Die Subjekte bzw. Ressourcen und Prädikate werden in RDF eindeutig gekennzeichnet, indem ihnen eine URI zugeordnet wird. Die URI dient lediglich zur eindeutigen Kennzeichnung der Ressourcen. Auch wenn diese **URI** oft die Form von

URLs annehmen, müssen sie nicht auflösbar sein oder Informationen bereit stellen, sondern dienen lediglich zur eindeutigen Kennzeichnung und Namensraumabgrenzung. Ein gültiges Beispiel ist im Folgenden dargestellt.

```
<http://linkedbuildingdata.net/ifc/resources20230612_174203/IfcPipeSegment_474>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://standards.buildingsmart.org/IFC/DEV/IFC4/ADD1/OWL#IfcPipeSegment>.
```

Lis. 7 gültiges RDF-Tripel serialisiert als RDF/XML (entspricht Tripel 5 aus Lis. 6)

Der wesentliche Vorteil einer Wissensrepräsentation mit Wissensgraphen (KG für engl. knowledge graph) besteht in einer einfachen Zusammenführbarkeit mehrerer Wissensquellen: Es ist lediglich die Vereinigungsmenge der beiden Teilmengen von Tripeln zu bilden. Identität wird aufgrund der weltweit einheitlichen Identifizierbarkeit mittels der URIs einfach erkannt.

Durch die einfache Zusammenführbarkeit unterschiedlicher Wissensgraphen verschwimmt die Grenze zwischen verschiedenen KG. Für die Abgrenzung eines Wissensgraphen zu einem anderen Wissensgraphen existiert keine allgemeinverbindliche Definition. Die Begrenzung ist abhängig von der genutzten Software-Implementierung zur Handhabung der KG. Denkbar wäre z.B. den Inhalt einer Datei oder einer Datenbank als einen KG zu begreifen.

2.3.2.2 Syntax

Für RDF wurden verschiedene Serialisierungen standardisiert, darunter RDF/XML, Notation3 und Turtle. In dieser Arbeit kommt ausschließlich die **Turtle-Syntax** (TTL) zum Einsatz, da es eine besonders gut menschenlesbare Serialisierung darstellt. In Turtle werden Tripel durch Punkte abgeschlossen. Die Abtrennung zwischen Subjekt, Prädikat und Objekt erfolgt mit Whitespaces. Für eine bessere Lesbarkeit können die URIs abgekürzt dargestellt werden. Dafür werden die URIs in Namespace und lokalen Bezeichner aufgeteilt. Für den Namespace wird einmalig ein Präfix definiert der innerhalb der Datei gültig ist. Das folgende Listing enthält dieselben Information wie Lis. 7, jedoch in dieser besser lesbaren Schreibweise.

```
@prefix inst: <http://linkedbuildingdata.net/ifc/resources20230612_174203/> .
@prefix rdf: <http://www.w3.org/2000/01/rdf#> .
@prefix ifc: <http://standards.buildingsmart.org/IFC/DEV/IFC4/ADD1/OWL#> .

inst: IfcPipeSegment_474 rdf:type ifc: IfcPipeSegment
```

Lis. 8 RDF-Tripel serialisiert als Turtle-Syntax (entspricht Tripel 5 aus Lis. 6)

2.3.2.3 Inhalte

RDF wurde ursprünglich als Beschreibungssprache für Meta-Informationen entwickelt, kann jedoch auch für jede andere Art von Informationen verwendet werden. Es können damit also sowohl Informationen aus dem Bereich des Instanzwissens, als auch aus dem Bereich des schematischen Wissens gespeichert werden. Im Kontext der Wissensgraphen sind dafür die Begriffe **A-Box** (assertional knowledge - Instanzwissen) und **T-Box** (terminological knowledge - Schemawissen) verbreitet. Lis. 9 zeigt dazu Beispiele aus der IFC-Domäne.

```
inst:IfcPipeSegment_474 ifc:objectPlacement IfcProduct ifc:IfcLocalPlacement_3527.
ifc:IfcPipeSegment rdfs:subClassOf ifc:IfcFlowSegment.
ifc:IfcPipeSegment owl:disjointWith ifc:IfcDuctSegment.
inst:IfcPipeSegment_474 rdf:type ifc:IfcPipeSegment
```

Lis. 9 Beispieltriple aus A-Box, T-Box und der Verbindung von A- und T-Box in Turtle-Syntax

Orange: Instanzwissen (A-Box), blau: Schemawissen (T-Box), pink: Verbindung

2.3.2.4 Abgrenzung zu alternativen Implementierungen

Für Wissensgraphen existieren alternative Implementierungen zu RDF, insb. sog. *Labelled Property Graphs* wie sie z.B. bei *neo4j* genutzt werden. Deren Knoten weisen im Gegensatz zu RDF-Knoten eine interne Struktur auf und erlauben insofern keine strenge Unterscheidung zwischen A-Box und

T-Box. Sie werden daher in dieser Arbeit nicht weiter betrachtet, stattdessen werden *Wissensgraphen* ausschließlich in Form von *RDF-Wissensgraphen* benutzt, daher werden beiden Begriffe im Folgenden synonym benutzt.

2.3.2.5 Triplestores

Wissensgraphen können in Dateien oder Datenbanken abgespeichert werden. Speziell dafür gemachte Datenbanken heißen *Graphdatenbanken* oder *Triplestores*. Die Zugriffsrechte werden dementsprechend auf Dateiebene oder von den Triplestores verwaltet. Von ganz restriktiv bis hin zu verfügbar für jedermann über das Internet ist dabei die ganze Skala realisierbar. Zur Verwaltung von Daten sind Kombinationen von Triplestores und klassischen Datenbanken möglich und oft sinnvoll.

Für die Implementierung in dieser Arbeit ist die Verwendung von Dateien für das Abspeichern der Wissensgraphen ausreichend, Triplestores kommen nicht zur Anwendung.

2.3.2.6 Abfrage und Manipulation von KG mit SPARQL

SPARQL ist eine graph-basierte Abfragesprache für RDF-basierte Datenquellen. Es existieren die Spezifikationen SPARQL-query sowie SPARQL-update, welche eine ähnliche Syntax verwenden. Lis. 10 zeigt ein Beispiel für eine SPARQL-Query. Angewandt auf den Wissensgraph in Lis. 9 würde sich `?rohrstueck=IfcPipeSegment_474` als Output ergeben.

```
SELECT ?rohrstueck
WHERE { ?rohrstueck rdf:type ifc:IfcPipeSegment. }
```

Lis. 10 Beispiel für eine SPARQL query

Wichtige Arten von Queries sind *SELECT*, *CONSTRUCT*, *INSERT*, *DELETE*, d.h. die Datenquelle kann mittels SPARQL nicht nur abgefragt, sondern auch manipuliert werden.

Von besonderer Bedeutung für die Abfrage von Wissensgraphen sind die sog. **PropertyChainAxioms**. Lis. 11 zeigt ein Beispiel. Es ist damit möglich Ketten von Property-Zuordnungen abzufragen. Diese können können von beliebiger Länge sein.

Beispieltriple	inst:IfcPipeSegment_474 ifc:objectPlacement_IfcProduct ifc:IfcLocalPlacement_3527 ifc:IfcLocalPlacement_3527 ifc:relativePlacement_IfcLocalPlacement inst:IfcAxis2Placement3D_111.
Mögliche Abfrage	SELECT ?x ?y WHERE { ?x ifc:objectPlacement_IfcProduct/ifc:relativePlacement_IfcLocalPlacement ?y }
Ergebnis	?x = inst:IfcPipeSegment_474 ?y = inst:IfcAxis2Placement3D_111

Lis. 11 Beispiel für ein PropertyChainAxiom

2.3.2.7 Einordnung verwandter Begriffe

Für die Verbindung mehrerer Wissensquellen unter Verwendung von RDF-Wissensgraphen wird der Begriff **Linked Data** verwendet. Befinden sich die Wissensquelle auf unterschiedlichen Rechnern, ist dafür der Begriff **Semantic Web** gebräuchlich. Abb. 30 zeigt die sog. „Linked Open Data Cloud“. Jeder dargestellte Punkt entspricht einem Datensatz im Umfang von min. 1000 Tripeln. Man erkennt den stetig wachsenden Informationsumfang, welcher durch Veröffentlichungen nach den *Linked Open Data Principles* zur Verfügung steht. RDF-Wissensgraphen bilden dafür die entscheidende technologische Grundlage.

2 Methodische und Technische Grundlagen

2.3 Semantic Technologies

2.3.3 Ontologies

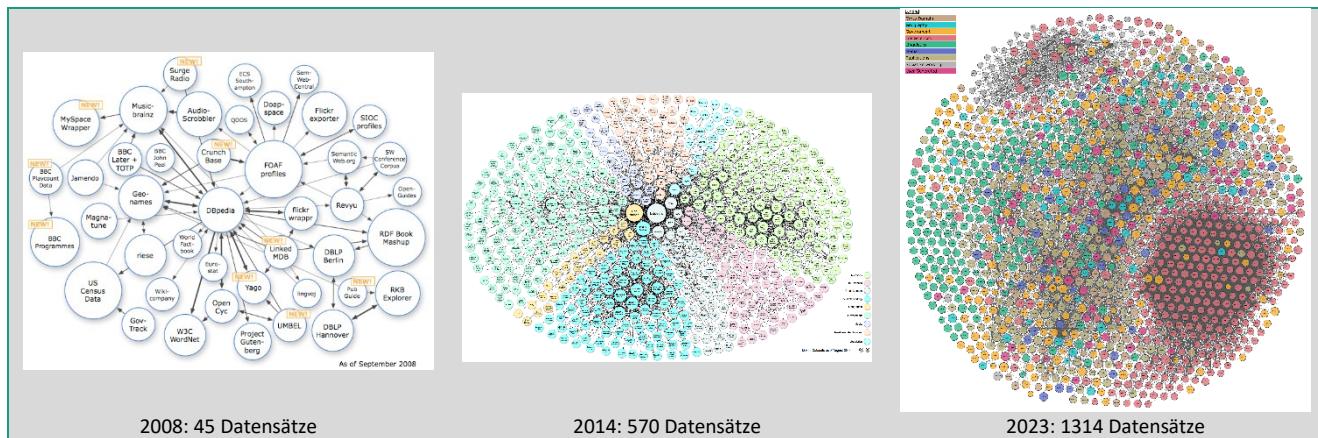


Abb. 30 Linked Open Data Cloud, Bildquelle: <https://lod-cloud.net/versions/>

Als **Knowledge Engineering** bezeichnet man den Prozess, Wissen in Form von RDF-Wissensgraphen zu formulieren, indem man passende Ontologien/Schemata sucht bzw. erstellt und die Daten dann mit den gefundenen Konzepten/Begriffen formuliert.

2.3.3 Ontologies

2.3.3.1 Begriff Ontologie

Der Begriff **Ontologie** stammt aus der Philosophie, wird aber auch in der Informatik verwendet und beschreibt für ein bestimmtes Themenfeld die dort vorkommenden **Begriffe²⁶ (Vokabular)**, die Zusammenhänge dieser Begriffe untereinander (**Terminologie**), sowie deren Bedeutung (**Semantik**). Tab. 25 zeigt ein Minimalbeispiel dazu.

Tab. 25 Minimalbeispiel für eine Gebäudeontologie

Vokabular	Gebäude Nichtwohngebäude (NWG) Wohngebäude (WG)
Terminologie	Gebäude ist der Oberbegriff zu NWG, WG. Ein Gebäude kann nur entweder als WG oder NWG eingestuft werden.
Bedeutung	Als Wohngebäude zählt jedes Gebäude, das mehr als 50% Fläche zur Wohnnutzung aufweist.

Eine Ontologie kann alternativ auch als „**Schema**“ bezeichnet werden. Sie enthält abstraktes Wissen. Damit können Daten/Informationen über konkrete Anwendungsfälle beschrieben werden. Ein Datenbank"schema" ist z.B. eine konkrete Umsetzung einer Ontologie. Darin ist festgelegt, welche Tabellen es gibt, welche Spalten mit welchem Wertebereich vorgesehen sind und was die Schlüssel sind anhand derer Tabellen miteinander verbunden werden. Das „Instanzwissen“ ist der Inhalt der Tabellen. Sowohl für die Daten, als auch für die Schemainformationen gibt es andere Darstellungsformen als die tabellarische – eine davon sind Wissensgraphen. Ontologien entsprechend damit der in Abschnitt 2.3.2.3 bereits thematisierten **T-Box**.

2.3.3.2 Basisontologien des W3C

Die Beschreibung einer solchen Ontologie muss in der Informatik computerlesbar d.h. formal erfolgen. Dafür sind z.B. Wissensgraphen geeignet. Für die Beschreibung insb. der Zusammenhänge sollte ein vordefiniertes Vokabular verwendet werden. Dieses wurde in sog. Basisontologien ebenfalls vom W3C standardisiert. Diese Definition liegen in Form von RDF vor. Wesentliche Ontologien und ihre enthaltenen Konzepte zeigt Tab. 26

²⁶ in dem Zusammenhang oft als „Konzepte“ bezeichnet

Tab. 26 Basisontologien

Basisontologie	Wichtige Konzepte	Standardnamespace	W3C Spezifikation
Resource Description Framework RDF	Rdf:type	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF 1.1 https://www.w3.org/TR/rdf11-concepts/
RDF Schema RDFS	Rdfs:domain Rdfs:range Rdfs:subPropertyOf Rdfs:subClassOf	http://www.w3.org/2000/01/rdf-schema#	-
Web Ontology Language OWL	Owl:equivalentClass Owl:disjointWith Owl:NamedIndividual Owl:ObjectProperty Owl:DatatypeProperty Owl:UnionOf Owl:IntersectionOf Owl:Restriction Owl:hasValue	http://www.w3.org/2002/07/owl#	OWL 2 https://www.w3.org/TR/owl2-overview/

2.3.3.3 OWL und Reasoning

OWL (Web Ontology Language) ist eine Ontologie für das Semantic Web. Die derzeit aktuelle Version OWL 2 wurde 2012 als W3C Empfehlung (OWL2) veröffentlicht. Mit OWL kann Wissen in einer der Prädikatenlogik ähnlichen Form ausgedrückt werden. Dies erlaubt Schlussfolgerungen zu ziehen. Tab. 26 listet ausgewählte Konzepte von OWL auf. Im Kontext dieser Arbeit werden hauptsächlich *owl:equivalentClass*, *owl:ObjectProperty* und *owl:DatatypeProperty* verwendet. Lis. 12 zeigt ein einfaches Beispiel für eine mögliche Schlussfolgerung.

```
inst:pump_1234 rdf:type aix:AixLib.Fluid.Movers.SpeedControlled_y.
aix:AixLib.Fluid.Movers.SpeedControlled_y owl:equivalentClass ifc:IfcPump.
Schlussfolgerung:
inst:pump_1234 rdf:type ifc:IfcPump.
```

Lis. 12 Beispiel Reasoning

Für das Schlussfolgern (im Folgenden: „Reasoning“ oder „**Semantic Reasoning**“) werden sog. **Reasoner** verwendet. Dabei handelt es sich um Softwaretools, welche als Bibliotheken/Plugins oder Standalone-Programm zur Verfügung stehen. Bekannte Vertreter sind HermiT, Pellet, FaCT++. Reasoner unterscheiden sich, hinsichtlich der damit handhabbaren OWL 2 Profile und ihrer Verfügbarkeit. Die Laufzeit und der Speicherbedarf eines solchen Reasoning hängen wesentlich vom verwendeten OWL Profil (siehe Tab. 28) ab.

Tab. 27 Reasoner – ergänzt nach (Noll 2013; „OWL Implementations“ 2020)

	Unterstützte OWL-Profile	Verfügbarkeit	Native Implementierung/Quelle
FaCT++	OWL DL	Protege-Plugin	C++
HermiT	OWL DL	Protege-Plugin OWLReady2 Jena	Java (Glimm u. a. 2014)
Pellet	OWL DL, EL	Protege-Plugin OWLReady2 Jena	Java

Für OWL wurde sowohl der volle Sprachumfang definiert, welcher als „OWL Full“ oder „OWL DL“ (Description Logic) bezeichnet wird, als auch sog. Profile. Die Profile lassen jeweils nur eine eingeschränkte Teilmenge des Sprachumfangs zu. Definiert sind drei solche Profile in OWL 2, sowie OWL Lite, welches bereits in OWL Version 1 spezifiziert wurde. Mit dem zulässigen Sprachumfang, sowie ggf. weiteren Restriktionen der Profile geht die Laufzeit der Reasoner einher.

2 Methodische und Technische Grundlagen

2.3 Semantische Technologien

2.3.3 Ontologien

Tab. 28 OWL Profile (übersetzt nach „OWL 2 Referenz“ 2012; „OWL Referenz“ 2004)

Bezeichnung	Reasoning	Anwendungsfall
OWL 2 EL EL family of description logics [EL++]	Basisreasoning ist möglich Polynomiale Laufzeit (bezogen auf die Größe der Ontologie)	Für Ontologien mit einer großen Anzahl von Eigenschaften und/oder Klassen
OWL 2 QL (Query Language)	Wenn die Beantwortung von Queries die wichtigste Reasoning-Aufgabe ist, kann eine Laufzeit im LOGSPACE erreicht werden Wie in OWL 2 EL können Algorithmen mit polynomieller Laufzeit verwendet werden für Ontologie-Konsistenzprüfungen und Subsumption von Klassenausdrücken Die Expressivität des Profil ist notwendigerweise begrenzt	Sehr großer Umfang an Instanzdaten
OWL 2 RL (Rule Language)	kann mit regelbasierten Reasonern implementiert werden. Konsistenzprüfung der Ontologie, Erfüllbarkeit von Klassenausdrücken, Subsumtion von Klassenausdrücken, Instanzprüfung und konjunktiver Anfragen mit polynomieller Laufzeit	Für Anwendungen mit skalierbarem Reasoning ohne zu viel Ausdruckskraft zu opfern
OWL Lite (Profil aus OWL 1)	geringere Komplexitätsklasse, daher schnellere Laufzeit	Klassenhierarchien können gebildet werden, Restriktionen auf Eigenschaften sind möglich, optionale und verpflichtende Eigenschaften sind möglich

Ontologien enthalten nach der o.g. Definition schematisches d.h. abstraktes Wissen. Sie dienen damit als Grundlage um Instanzwissen formal korrekt beschreiben zu können. Da die Basis-Ontologien des W3C als RDF-Wissensgraphen vorliegen und diese besonders gut zusammenführbar sind, hat sich RDF auch für die Definition von Domänen-Ontologien durchgesetzt und wird auch in dieser Arbeit verwendet.

2.3.3.4 Domänen-Ontologien

Domänen-Ontologien nutzen i.d.R. die beschriebenen Basisontologien als Grundlage, wie die Prinzipdarstellung in Abb. 31 zeigt. Beispiele für Domänenontologien wurden in Abschnitt 1.2.3.3 für Anlagen- und Regelungstechnik, sowie im Abschnitt 1.4.3 für die Modelica-Domäne genannt. Von großer Bedeutung für diese Arbeit in die IFC-OWL-Ontologie, welche das IFC-Schema, wie es in Abschnitt 2.1 beschrieben wurde, widergibt. Neben EXPRESS wird das IFC-Schema von buildingSMART auch als OWL-Ontologie bereitgestellt.

2.3.3.5 Abgrenzung Ontologie und Wissensgraph

Es wurde oben bereits beschrieben, dass mit RDF-Wissensgraphen sowohl Instanz- als auch Schemawissen dargestellt werden kann. Beides wird in der Praxis verwendet. In der Linked-Data-Community wird der Begriff *Ontologie* oft synonym für RDF-Wissensgraphen verwendet, d.h. sowohl für Graphen, die schematisches Wissen enthalten, als auch für solche, welche Instanzwissen enthalten. Dies ist nachvollziehbar, da beides technisch nicht auseinander gehalten werden kann, jedoch wird dieser irreführende Sprachgebrauch in dieser Arbeit NICHT verwendet. Als *Ontologie* wird lediglich der Wissensgraph zur Repräsentation des schematischen Wissens bezeichnet. Eine Knowledge-Base (KB) besteht aus ein oder mehreren Ontologien (T-Box) und/oder ein oder mehreren Instanzgraphen (A-Box). Instanzwissen kann i.d.R. nur mit einem bestimmten Vokabular ausgedrückt werden, daher bauen Instanz-KG auf Domänenontologien auf, dieser wiederum nutzen Basisontologien, um ihre Inhalte auszudrücken. Das Konzept dieses Wissensgraphen-Stapels wird in Abb. 31 veranschaulicht. Dabei ist auf der linken Seite ein Beispiel für ein Simulationsmodell *inst1* dargestellt, auf der rechten Seite ist ein IFC-Modell *inst2* dargestellt. Beide nutzen dieselben Basisontologien, jedoch unterschiedliche Domänenontologien. Die Modelica-seitigen Ontologien wurden im Kontext dieser Arbeit erstellt und werden in den Abschnitten 3.6 und 3.7 erläutert.

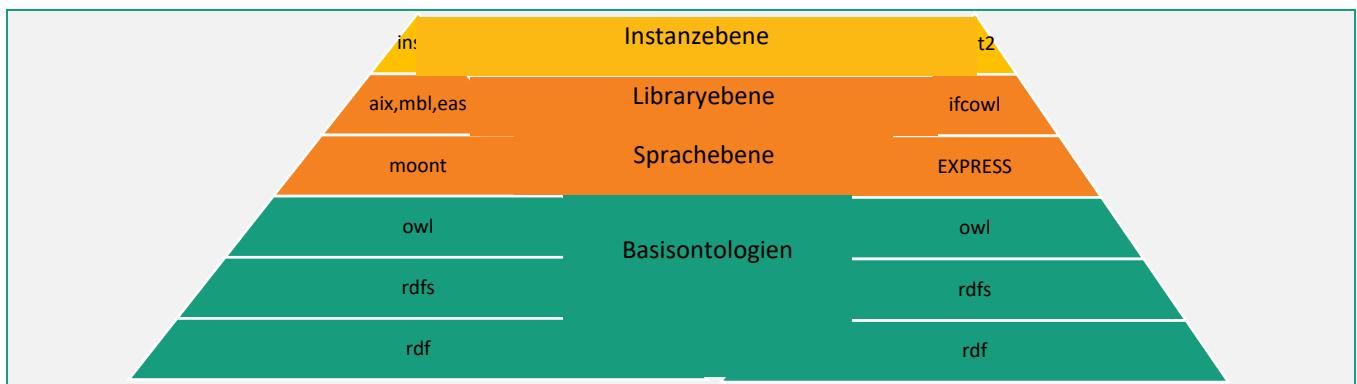


Abb. 31 Wissensgraphen-Stapel für die Domäne Modelica (links) und IFC (rechts) (orange: Domänen-Ontologien, gelb: Instanz-KG)

2.3.4 Tools zum Handling von Wissensgraphen

Im Verlauf der vergangenen 30 Jahre wurden mit der Entwicklung des Semantic Web auch zahlreiche Tools zur Handhabung von RDF-Wissensgraphen entwickelt. Im Folgenden sollen die im Zusammenhang mit dieser Arbeit verwendeten Werkzeuge kurz beschrieben werden.

2.3.4.1 Apache Jena

Apache Jena (siehe Tab. 54) ist ein Java-Framework zur Handhabung von Wissensgraphen incl. Ontologien. Es wird seit 2011 von der Apache Foundation gepflegt und liegt als FLOSS-Software vor. Enthalten ist eine Implementierung von SPARQL, die Triplestores TDB und Fuseki, die Handhabung von RDF-Wissensgraphen, sowie verschiedene Reasoner.

2.3.4.2 Protege

Protégé ist ein weiterverbreiteter FLOSS-Ontologie Editor (siehe Tab. 54). Er steht als Rich-Client für verschiedene Betriebssysteme zur Verfügung, sowie inzwischen auch als Webanwendung. Protege ermöglicht das Bearbeiten allgemeiner Wissensgraphen – nicht nur von Ontologien, wie die Bezeichnung vermuten lässt (zur Begriffs differenzierung siehe S. 90). Protege arbeitet vorrangig mit Wissensgraphen, welche in Dateien vorliegen, bietet jedoch auch eine Benutzeroberfläche, um entfernte Triplestores anzusprechen. Enthalten sind weiterhin Implementierungen für Reasoner und Visualisierungen für Wissensgraphen.

2.3.4.3 Rdflib

RDFLib ist eine Pythonbibliothek zur Handhabung von RDF-Wissensgraphen (siehe Tab. 54). Sie ermöglicht das Einlesen und Serialisieren von RDF-Graphen in verschiedenen Formaten, darunter das o.g. Turtle-Format, außerdem können Triplestores damit angebunden werden. Implementierungen für SPARQL-Query und SPARQL-Update sind ebenfalls enthalten.

2.3.4.4 Owlready2

OWLReady2 (siehe Tab. 54) ist eine Pythonbibliothek für die Handhabung von Ontologien und unterscheidet sich von der RDFLib insb. durch das Vorhandensein von Reasonern. OWLReady2 kann OWL2-Ontologien (siehe dazu Tab. 28 OWL Profile) verarbeiten und kann die Reasoner HermiT und Pellet nutzen.

3 Methodischer Ansatz

Es wird zunächst der grundsätzliche Lösungsansatz für die Übersetzung zwischen IFC und Modelica dargestellt. Wissensgraphen spielen eine zentrale Rolle für den Lösungsansatz, deren Vorteile gegenüber anderen Ansätzen werden präsentiert. Anschließend werden die für die Anwendung auf die beiden UseCases BIM2SIM und SIM2BIM notwendigen Softwarekomponenten (Konverter) abgeleitet. Es ergeben sich drei Gruppen von notwendigen Konvertern: Pre-/Postprocessing, Translatoren, Transkriptoren. Die Grundkonzepte der einzelnen Konverter werden anschließend dargestellt (3.5, 3.8, 3.9, 3.10, 3.11, 3.12).

Für die Transkription von Modelica-Modellen in Wissensgraphen ist es nötig, zunächst eine Modelica-Ontologie zu definieren. Deren Inhalte und die Entwurfsentscheidungen werden erläutert (3.6). Es wird der MoTTL-Transcriptor beschrieben, welcher implementiert wurde, um Modelica-Modelle mit Hilfe der Modelica-Ontologie in Wissensgraphen zu überführen (3.10). Mit dessen Hilfe wurden Ontologien der Modelica-Libraries erstellt (3.7), welche für die „Semantic Translation“ notwendig sind.

Für die Translate-Komponenten sind zwei verschiedene Arten der Implementierung – „Semantic“ und „Procedural Translation“ – denkbar. Beide (3.8, 3.11) werden mit ihren Vor- und Nachteilen beschrieben. Für die Semantic Translation sind sog. Alignments und die Library-Ontologien notwendig. Theoretische Grundlagen dazu, sowie das konkrete Alignment für die beiden UseCases dieser Arbeit werden dargestellt.

3.1 Lösungsarchitektur des MO-x-IFC-Toolset²⁷

Abb. 32 stellt drei Vorgehensweisen für die Erstellung von Anlagen-Simulationsmodellen aus BIM-Modellen gegenüber: Die manuelle Erzeugung von Simulationsmodellen aus IFC-Modellen mit Hilfe von Simulationsbibliotheken ist derzeit immer noch Stand der Technik bei der Anwendung in Ingenieurbüros. Die Nutzung von Konvertierungswerkzeugen unter Einbeziehung von Metamodellen ist Stand der Wissenschaft, wie es in Abschnitt 1.4.1 dargestellt wurde.

An die Stelle *eines* Konvertertools soll ein modulares Datenflusssystem²⁸ namens „MO-x-IFC-Toolset“ treten. Die notwendigen Prozessschritte erzeugen dabei jeweils aus einem „Input“ einen „Output“. Der „Output“ des vorigen Prozessschritts bildet dabei den Input des folgenden Prozessschritts. Die Zwischenstände zwischen zwei Prozessschritten werden als Zustände (ZS) bezeichnet.

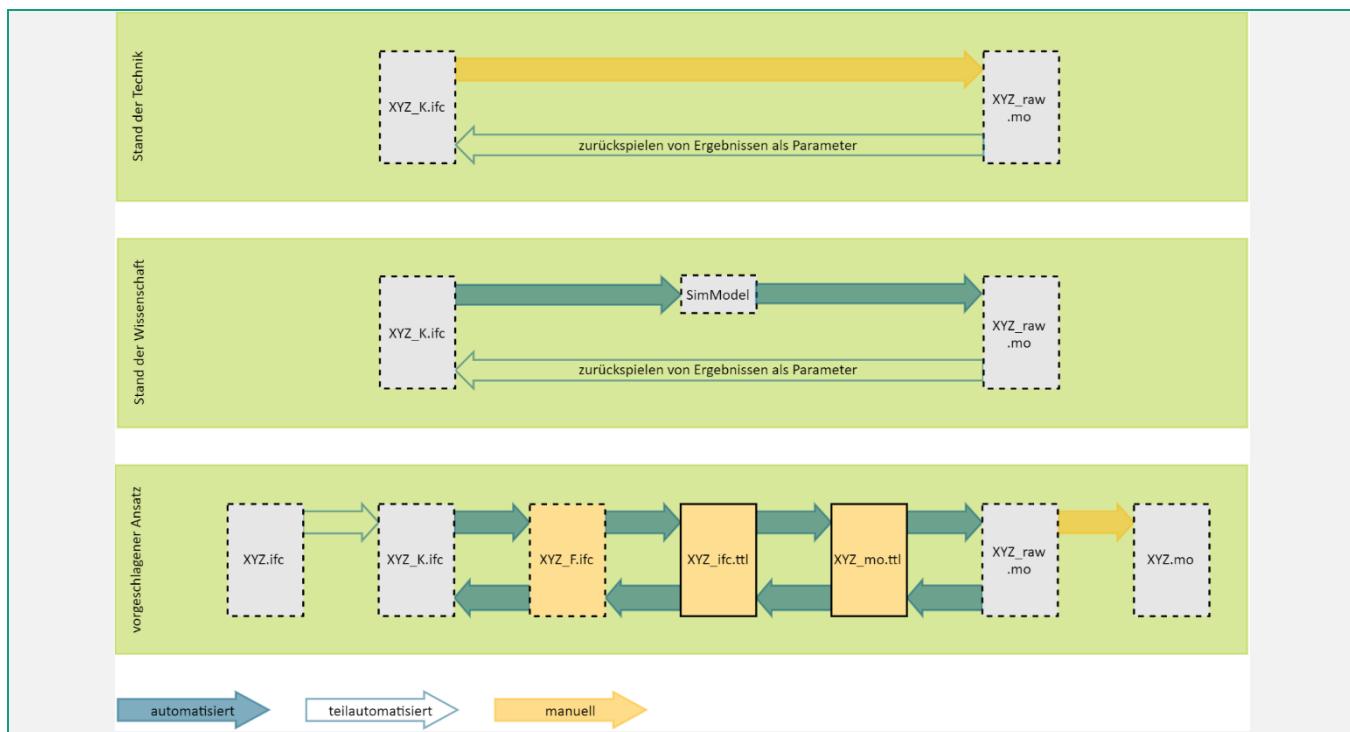


Abb. 32 Status Quo, klassischer und semantischer Lösungsansatz

Um eine modulare Anwendbarkeit der verschiedenen Prozessschritte zu gewährleisten, müssen alle ZS in Form standardisierter Dateien vorliegen: Außer den beiden nativen Dateiformaten IFC-STEP und Modelica, sollen dabei ausschließlich RDF-Wissensgraphen zum Einsatz kommen. Dies ist ein wesentlicher Unterschied zu den in der Literatur vorgestellten Konvertierungswerkzeugen, die oftmals auf speziell für den Anwendungsfall definierte Datenschemata und Dateiformate setzen. Die ZS sollen jeweils alle für die Folgeschritte notwendigen Informationen enthalten und sind in Abb. 33 dargestellt.

²⁷ Konzept bereits vorgestellt: (Eckstädt 2021)

²⁸ „pipes-and-filters“

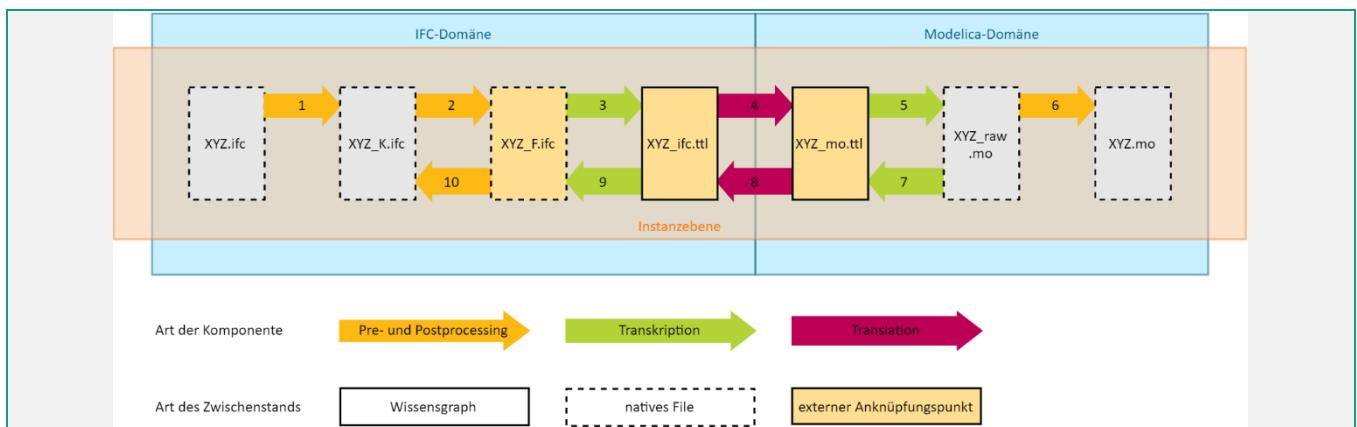


Abb. 33 Zusammenwirken von MO-x-IFC-Konverter (MOXIK) für die UC1 und UC2 zur Überführung zwischen IFC und Modelica
Erläuterung der Ziffern in Tab. 30

Die verschiedenen Prozessschritte des MO-x-IFC-Workflow werden im Folgenden auch als MO-x-IFC-Konverter (MOXIK) bezeichnet. Jeder dieser Konverter zeichnet sich durch einen definierten In- und Output aus. Konverter, welche zwischen zwei RDF-Wissensgraphen überführen, werden als **Translator** bezeichnet. Konverter, die die beteiligten nativen Dateiformate (IFC-STEP und Modelica) in Wissensgraphen übertragen, werden als **Transkriptoren** bezeichnet. Konverter, die die nativen Dateiformate behandeln, werden als Pre- bzw. Postprocessingkomponenten (PPP) bezeichnet.

Translator können mittels Semantic Reasoning (siehe Abschnitt 2.3.3.3) und Alignments bidirektional funktionieren. Sie werden dann als **Semantic Translator** bezeichnet. Alternativ kann die Überführung zwischen zwei RDF-Wissensgraphen klassisch richtungsabhängig implementiert werden und wird dann als **Procedural Translator** bezeichnet.

3.2 Vorteile durch die Verwendung von Wissensgraphen

Durch die Transkription des IFC- bzw. Modelica-Models in einen KG kann man Informationen verknüpfen und somit Wissen schaffen (siehe Abschnitt 2.3.1/ Abb. 29). Dies zeigt Abb. 34: Innerhalb der Wissensebene (symbolisiert durch die Wolke) können die verschiedenen Ontologien mithilfe sog. Alignments verbunden werden.

Ein Alignment enthält die Zuordnung von Entitäten in einer Ontologie A zu passenden Entitäten in einer anderen Ontologie B. Neben der Entsprechung *owl:equivalentClass* können auch andere Beziehungstypen, z.B. *rdfs:subClassOf* definiert werden. Weitere Möglichkeiten sind in Tab. 26 (S. 89) aufgeführt. Damit wird - zumindest für die Entsprechungen *owl:equivalentClass* - eine automatische Übersetzung mittels Reasoning möglich. Dieses Konzept wird in dieser Arbeit als **Semantic Translation** bezeichnet und in Abschnitt 3.7 näher erläutert.

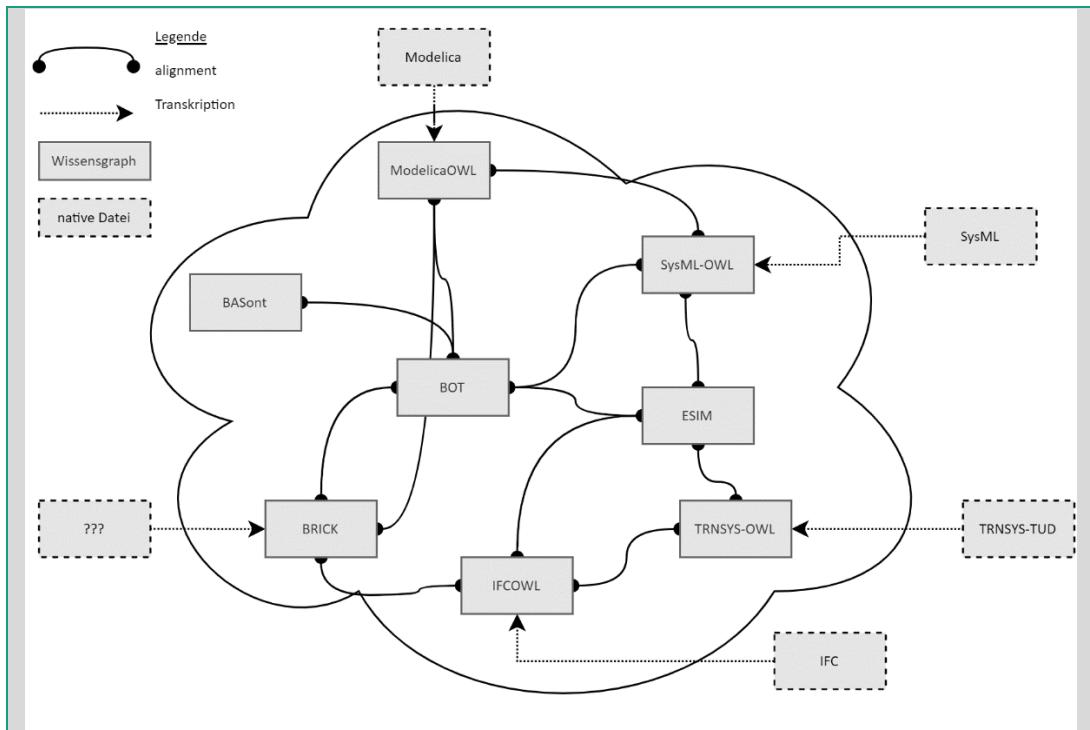


Abb. 34 Verknüpfung von Informationen zu Wissen mit Hilfe von Wissensgraphen

Die Grafik in Abb. 34 deutet an, dass dies nicht durch eine direkte Übersetzung erfolgen muss, sondern auch indirekt über andere Ontologien erfolgen kann. Dieser Vorteil kommt besonders zum Tragen, wenn man weitere Wissensquellen in die Überlegungen mit einbezieht (dargestellt als Beispiel BRICK). Um die Möglichkeiten zu illustrieren ist in Abb. 30 (S. 88) eine Visualisierung der „Linked Open Data Cloud“ dargestellt - man erkennt die Vielzahl der frei verfügbaren Ontologien. Die Übersetzung mittels Zwischenontologien kann jedoch auch Probleme verursachen, insbesondere, wenn sich die Definition einer dieser Zwischenontologien ändert. Daher wird ein solcher „Umweg“ nicht präferiert, kann jedoch bei vorhandenen Vorarbeiten sinnvoll sein.

Es wurden Wissensgraphen als Zwischenstände der Umwandlung zwischen Modelica und IFC gewählt, da diese folgende Vorteile bieten:

1. Die Überführung zwischen verschiedenen Wissensgraphen kann teilweise automatisch mit Hilfe von sog. Reasonern und Alignments erfolgen (siehe Abschnitt 3.8)
2. Wissensgraphen sind hochgradig interoperabel und erfahren eine zunehmende Anwendung im Semantic Web, aber auch innerhalb geschlossener Systeme.

Daraus ergeben sich weitere Anwendungsmöglichkeiten für die Zwischenschritte. Dies wird z.B. für den Mo-KG (`XYZ_mo.ttl` in Abb. 33) in (Eckstädt, Menzel, u. a. 2023), sowie darüber hinaus in Abschnitt 6.4.1 beschrieben.

Wissensgraphen sind aufgrund ihrer Interoperabilität auch kompatibel mit dem *big open BIM*-Konzept (siehe Abschnitt 1.1.3)

3. Es existieren bereits Tools um IFC-STEP-Dateien in oder aus IFC-OWL-Graphen zu erzeugen (Konverter 3 und 8 in Abb. 33), dadurch entsteht kein Mehraufwand bei der Verwendung von IFC-OWL statt IFC-STEP.

Weitere Vorteile die sich aus der Transkription von Modelica-Modellen in Wissensgraphen ergeben wurden bereits in Abschnitt 1.4.3 beschrieben.

3.3 Anforderungen an das zu entwickelnde Toolset

Das zu entwickelnde Toolset (bzw. einzelne Komponenten davon) soll sich bestmöglich in bestehende Arbeitsabläufe in der Planung integrieren. Die Zusammenarbeit insb. auch mit Akteuren, welche ohne Autorentool (siehe Abschnitt 1.2.2.1) arbeiten, soll ermöglicht bzw. verbessert werden. Daher werden für die Entwicklung des MO-x-IFC-Toolsets folgende Anforderungen aufgestellt:

1. Standard-Formate: Es sollen möglichst keine/wenige benutzerdefinierte Formate zur Anwendung kommen. Stattdessen sollen die umfangreichen Möglichkeiten des IFC-Standards, wie sie in Abschnitt 2.1 beschrieben wurden, zur Anwendung kommen. Dies ermöglicht die „nahtlose“ Zusammenarbeit mit marktverfügbarer kommerziellen oder offenen Softwaretools z.B. zur Visualisierung.
2. Standard-Bibliotheken: Auch Modelica-seitig soll möglichst auf die Definition eigener Bibliotheks-Komponenten verzichtet werden, stattdessen sollen Komponenten aus der MSL, sowie den Domänen-Libraries AixLib und MBL (siehe Abschnitt 2.2.6) verwendet werden. Diese sind open source verfügbar, aufgrund der vorhandenen großen Community und der bereits investierten langjährigen Entwicklungsarbeit wird von einer guten Qualität dieser Modelle ausgegangen.

Da es jedoch Fälle gibt, in denen die Definition eigener Bibliotheks-Komponenten sinnvoll und nötig ist (z.B. andere Szenarien als die in Abschnitt 1.3.1 thematisierten), soll dieser Anwendungsfall bei der Entwicklung der Konverter berücksichtigt werden. Die dafür notwendigen Anpassungen werden im Ausblick (6.4.4) thematisiert, sowie exemplarisch in Abschnitt 3.8.4.3.

3.4 Überblick über die MO-x-IFC-Konverter

Abb. 33 zeigt das Zusammenwirken der MOXIK bei der Überführung von IFC-STEP in Modelica und von Modelica zu IFC-STEP. Sie umfasst also die beiden in Abschnitt 1.5.1 definierten Usecases (UC1, UC2). Man erkennt die beiden Dateiformate IFC und Modelica an beiden Enden der Kette. Diese wurden in den Abschnitten 2.1 und 2.2 detailliert beschrieben. Dargestellt sind neben den „Zwischenständen“ (ZS) die MOXIK als Pfeile. Abb. 35 zeigt den Ausschnitt des Übersetzungsprozess, der in der semantischen Domäne abläuft. Die auf schematischer Ebene beteiligten Wissensgraphen sind zusätzlich dargestellt.

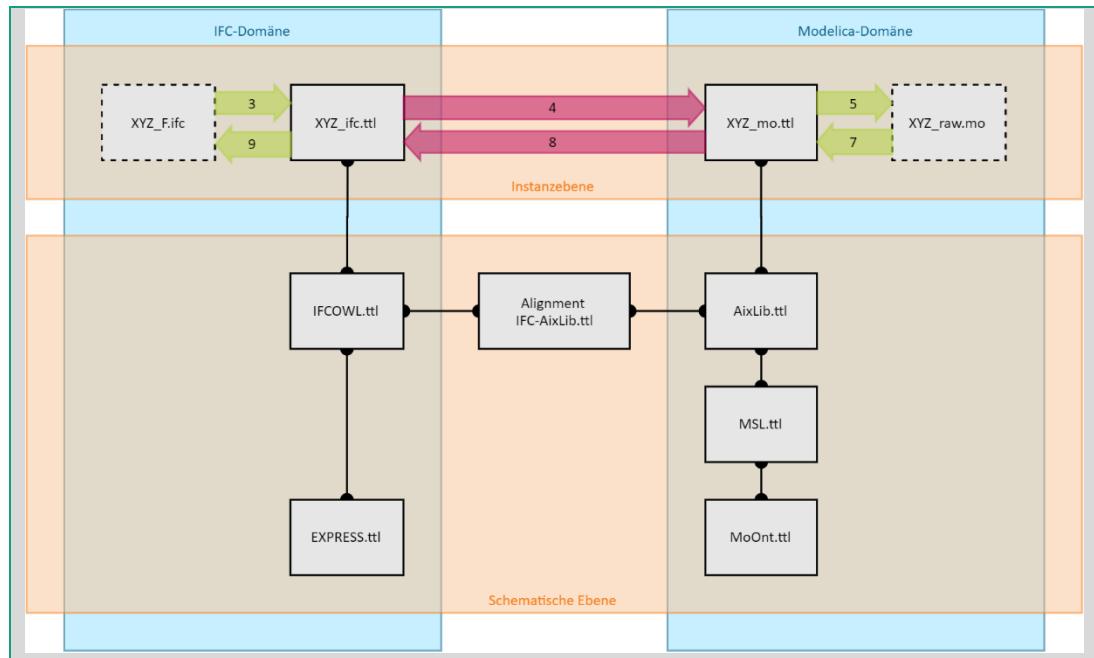


Abb. 35 beteiligte Wissensgraphen für die bidirektionale Übersetzung

Tab. 29 fasst alle ZS des MO-x-IFC-Workflows zur Überführung zwischen IFC-STEP und Modelica zusammen. Es handelt sich dabei um verschiedene Modelle ein und desselben realen Objekts – im Falle dieser Arbeit: Anlagen(teile) der technischen Gebäudeausrüstung. Die Herangehensweise lässt sich jedoch auf andere Objekte übertragen, siehe dazu Abschnitt 6.4 auf S. 176.

Wesentliche Zwischenschritte sind die beiden gelb hervorgehobene RDF-Wissensgraphen (`XYZ_ifc.ttl` und `XYZ_mo.ttl`), zwischen denen die sog. *Translation* stattfindet. Beide zeichnen sich durch unterschiedliches Vokabular aus, wie Abb. 35 zeigt. Die Grundlagen dazu wurden in Abschnitt 2.3.2 und 2.3.3 beschrieben. Ausführungen zu den konkret verwendeten Vokabularen IFC-OWL und MoOnt finden sich in den Abschnitten 2.1.2 und 3.6.

Tab. 29 Zwischenstände bei Anwendung von MO-x-IFC für die UC1 und UC2

Dateiname und Format	Kurzbezeichnung	Dateiinhalt	Identifikator der Dateikomponenten
XYZ.ifc	Ursprungsmodell	Ähnlich dem Konstruktionsmodell, jedoch ohne Erfüllung der formulierten Anforderungen	Instanz-ID GlobalId
XYZ_K.ifc IFC STEP	Konstruktionsmodell	entsprechend der in Abschnitt 3.5.1 formulierten Anforderungen: 3D-Geometrie der Anlagenkomponenten, Informationen zur logischen Verknüpfung, wesentliche Eigenschaften als Attribute oder Properties	Instanz-ID GlobalId
XYZ_F.ifc IFC STEP	IFC-Funktionsmodell	berechnungsrelevante Objekte (Replacements und strukturrelevante Bauteile), Informationen zu deren logischen Verknüpfung, wesentliche Eigenschaften als Attribute oder Properties	Instanz-ID GlobalId
XYZ_ifc.ttl TTL mit IFCOWL-Vokabular	IFC-KG bzw. IFC-KG-vollst	wie XYZ_F.ifc	URI
XYZ_classes_ifc.ttl* TTL mit IFCOWL-Vokabular	IFC-KG-classes	wie XYZ_F.ifc	URI
XYZ_classes_mo.ttl* TTL mit MoOnt-Vokabular	Mo-KG-classes	wie XYZ_F.ifc	URI
XYZ_mo.ttl TTL mit MoOnt-Vokabular	Mo-KG bzw. Mo-KG-vollst	wie XYZ_F.ifc	URI
XYZ_raw.mo Modelica-nativ	Strukturmodell	wie XYZ_F.ifc	Pfad im Package Komponentenname
XYZ.mo Modelica-nativ	Simulationsmodell	wie XYZ_F.ifc	Pfad im Package Komponentenname

*diese internen Zwischenstände der Konverter 4 bzw. 8 haben keine externe Anknüpfungspunkte und sind daher in Abb. 33 nicht dargestellt. Ihre Einordnung erfolgt in Abschnitt 3.8 / Abb. 39 bzw. 3.11 / Abb. 45.

Die im weiteren Verlauf der Arbeit genutzten Bezeichnungen für die verschiedenen Zwischenstände sind in Tab. 29 in Spalte 1 und 2 aufgelistet. In dieser Arbeit wird davon ausgegangen, dass jeder ZS als Datei vorliegt. Das entsprechende Format ist in Tab. 29 aufgeführt. Die Herangehensweise ist jedoch auch gültig, wenn statt Dateien Datenbanken verwendet würden.

Die Modelle unterscheiden sich in ihrem Format, welches an der Dateiendung erkennbar ist. Es werden zusätzliche Suffixe verwendet, um die Dateien eindeutig zu bezeichnen. In jedem Dateiformat ist ein eindeutiger Identifikator für die einzelnen Modellkomponenten vorhanden, welche in der letzten Spalte der Tab. 29 genannt ist. Diese Identifikatoren bei allen Modelltransformationen zu erhalten ist eine wesentliche Anforderung an die Implementierung aller Konverter, um die bidirektionale Anwendbarkeit des Workflows zu gewährleisten.

Abb. 33 zeigt, dass in der IFC-Domäne zwei STEP-Files beteiligt sind. Diese bilden zwei wesentliche Informationsmodelle in der Planung ab:

- Das Konstruktionsmodell entspricht der Datei **XYZ_K.ifc** (Beispiel in Abb. 51 auf S. 146),
- das Funktionsmodell entspricht der Datei **XYZ_F.ifc** (Beispiel in Abb. 50 auf S. 145).

Bereits aus der Umwandlung dieser beiden Darstellungsformen ineinander entsteht erheblicher Mehrwert. Dieser wird in Abschnitt 1.1.2 erläutert und entspricht den gelb hervorgehobenen Pfeilen in Abb. 6 und Abb. 8. Die zugehörigen Konverter werden in den Abschnitten 3.5 und 3.12 erläutert.

Wie in Abschnitt 2.1 beschrieben wurde, gibt es oftmals mehrere IFC-Schema-konforme Varianten, um bestimmte Informationen in IFC darzustellen. Darüber hinaus exportieren viele Autorentools sogar nicht-schemakonforme Dateien. Daher ist es notwendig, die zu verwendenden IFC-Dateien auf Konformität mit den Anforderungen für den Workflow in dieser Arbeit zu prüfen. Für den Fall, dass diese nicht erfüllt sind, sind in Abschnitt 4.3.1 Möglichkeiten und Tools beschrieben, um vorhandene IFC-STEP-Dateien so nachzuarbeiten, dass sie für den MO-x-IFC-Workflow verwendbar sind. Der Pfeil 1 in Abb. 33 repräsentiert diese Teile des Workflows.

Für die Transkription von IFC-STEP (2D oder 3D) in IFC-KG und umgekehrt existieren Open-Source-Komponenten *IfctoRDF* und *IfcOWL-to-IfcSTEP* (siehe Tab. 54) welche genutzt werden können. Sie sind in Abb. 35 mit Nummer 3 und 9 gekennzeichnet. Für die Transkription zwischen den ZS *XYZ_mo.ttl* und *XYZ_raw.mo* werden eigene Komponenten entwickelt, dieser sog. MoTTL-Transcriptor bzw. MoTTL-Descriptor werden in den Abschnitten 3.8.4 und 3.10 beschrieben.

Der ZS *XYZ_raw.mo* entspricht dem in Abschnitt 1.5.2 beschriebenen Modelica-Modell mit valider Modellstruktur und geeigneter Vorparametrierung. Er dient der manuellen Weiterbearbeitung, um daraus ein rechenfähiges und sinnvoll parametriertes Simulationsmodell zu erstellen.

Für den Prozessschritt *Translation* wurden zwei unterschiedliche Konzepte untersucht:

- Variante A: *Semantic Translator* und
- Variante B: *Procedural Translator*.

Diese werden in den Abschnitten 3.7 und 3.11 erläutert.

Tab. 30 zeigt eine Übersicht der beteiligten Konverter.

Tab. 30 Übersicht MO-x-IFC-Konverter (MOXIK)

Nr	Kurzbezeichnung	Inhalt	Verweis zur Erläuterung der Methodik
Pre- und Postprocessing			
1		Anforderungen an IFC erfüllen	3.5.1
2	Schematizer	Schematisierer	3.5
		Vereinfachung Modell	3.5.3
		Platzierung 2D	3.5.4
		Abbildung in IFC	2.1.10
10	Voluminizer	Ergänzung von (Platzhalter)-Geometrien (2D, 3D)	3.12
Transkription			
3	IFC2RDF	Überführung IFC in KG	-
9	RDF2IFC	Überführung von KG in IFC-STEP	-
5	TTL2MO		3.9
7	MOTTL		3.10
Translate			
4	SemTran	Semantic Translation	3.8
8	ProTran	Procedural Translation	3.11

3.5 Schematizer (Konverter 2)

In Abschnitt 1.1.2 wurde bereits herausgearbeitet, dass Funktionsmodelle (und die zugehörige Dokumentart Schema) für viele Planungsschritte besser geeignet sind als Konstruktionsmodelle. Dies gilt auch für die Verwendung zur Anlagensimulation. Hier kommt besonders der Aspekt zum Tragen, dass Funktionsmodelle deutlich weniger Komponenten enthalten als Konstruktionsmodelle. Bei einer AS sind für jede enthaltene Komponente Gleichungen zu lösen. Eine höhere Anzahl von Gleichungen sorgt für eine mindestens um denselben Faktor ggf. auch überproportional erhöhte Rechenzeit, daher sollten nur die simulationsrelevanten Komponenten enthalten sein. Der Konverter Schematizer soll ein Funktionsmodell automatisiert aus einem Konstruktionsmodell erstellen.

3.5.1 Anforderungen / Voraussetzungen der IFC-Files

Konstruktionsmodelle (IFC-STEP-Dateien), welche mit dem MO-x-IFC-Toolset verarbeitet werden sollen, müssen eine Reihe von Voraussetzungen erfüllen:

1. Einige der verwendeten Klassen sind erst in IFC4 definiert, IFC4.0.2.1 stellt daher die Mindestanforderung hinsichtlich der Dateiversion dar.

Für die in der Datei enthaltenen *IfcProducts* (d.h. alle „realen Objekte“ mit geometrischer und räumlicher Zuordnung) sind folgende Anforderungen zwingend:

2. Korrekte Klassenzuordnung: Tab. 31 zeigt die unterstützten Klassen. *IfcProducts* mit einer anderen Klassenzuordnung insb. *IfcBuildingElementProxy*, aber z.B. auch Elemente aus der Architekturdomäne wie *IfcWall* und *IfcWindow* werden ignoriert und vom MO-x-IFC-Toolset nicht behandelt.
3. Korrekte Typisierung: Die behandelten *IfcProduct* müssen via *IfcRelDefinesByType* ein Type-Objekt zugeordnet haben. Dessen Attribut *PredefinedType* muss gemäß Tab. 31 belegt sein. Andernfalls wird Modelica-seitig lediglich ein Platzhalterelement ohne inhaltlichen Mehrwert erzeugt.
4. Verbindungen: Bauteile müssen mit Hilfe von *IfcDistributionPorts* miteinander verbunden sein (Beschreibung dazu siehe Abschnitt 2.1.9.1). Die daraus abzulesenden Anlagengraphen sollen möglichst geschlossen sein.
5. Portbezeichnungen: *IfcDistributionPorts* müssen eine eindeutige und Modelica-kompatible Bezeichnung haben: Bei Elementen mit zwei hydraulischen Anschlüssen, müssen diese *port_a* und *port_b* heißen, andernfalls sollen die Ports durchnummiert sein: *port_1*, *port_2* und *port_3*.

Tab. 31 unterstützt Klassen und predefined Types im IFC-AixLib-Mapping

Klasse	Type	PredefinedType	Verwendung im Bsp.	Nicht-strukturrelevant (als Bestandteil eines Replacement)
IfcChiller	IfcChillerType	WATERCOOLED	Kältemaschine	
IfcCoil	IfcCoilType	HYDRONICCOIL	Serverschränke (2x) Rückkühlung(2x)	
		DXCOOLINGCOIL	Inneneinheit Splitgerät (2x)	
IfcFilter	IfcFilterType	WATERFILTER	Wasserfilter	x
IfcFlowInstrument	IfcFlowInstrumentType	PRESSUREGAUGE		
IfcHeatExchanger	IfcHeatExchangerType	PLATE	Wärmeübertrager (Systemtrennung BKA) (1x)	
		USERDEFINED	BKA Kreise, Heizkreise	
IfcPipeFitting	IfcPipeFittingType	BEND	Bögen	x
		TRANSITION	Übergangsstücke	x
		CONNECTOR	Flansch	x
		JUNCTION		
IfcPipeSegment	IfcPipeSegmentType	RIGIDSEGMENT	gerade Rohrstücke	x
IfcPump	IfcPumpType	CIRCULATOR	Umwälzpumpe	
IfcSensor	IfcSensorTYpe	FLOWSENSOR	Volumenstrommesser	
		HEATSENSOR	Wärmemengenzähler	
IfcTank	IfcTankType	EXPANSION	Druckhaltung MAG	
		PRESSUREVESSEL	Kaltwasserspeicher	
		STORAGE	-	
IfcUnitaryEquipment	IfcUnitaryEquipmentType	USERDEFINED	Entgasung (1x)	
		AIRCONDITIONINGUNIT	Lüftungsgeräte (4x)	
		SPLITSYSTEM	Split-Kühlgerät (2x)	
		AIRHANDLER	Umluftkühlgerät (43x)	
IfcValve	IfcValveType	CHECK	Rückschlagventile	
		ISOLATING	Absperrventil (255x)	x
		PRESSUREREDUCING	Strangregulierventile (71x)	x
		REGULATING	GA-aufgeschaltet (6x)	
		MIXING	3-Wege-Ventile (4x)	
IfcVirtualElement	IfcPipeSegmentType IfcDistributionElementType	USERDEFINED	Ersatzdruckverluste	

Um das Handling und die Interpretierbarkeit der Ergebnisse zu vereinfachen, sollen folgende Voraussetzungen für alle *IfcProducts* erfüllt sein:

5. Korrekte Zuordnung zu Geschossen (siehe Abschnitt 2.1.9.2)
6. Korrekte Zuordnung zu Systemen (siehe Abschnitt 2.1.9.3)
7. (Kurze) eindeutige menschenlesbare Namen der *IfcProducts*: Alle *IfcRoot*-Entitäten haben mit der GlobalID eine eindeutige maschinenlesbare Kennung. Jedoch sind diese aufgrund ihrer Beschaffenheit (siehe Abschnitt 2.1.4.1) für menschliche Nutzer schlecht handhabbar. Es ist hilfreich, wenn zusätzlich eine eindeutige menschenlesbare Bezeichnung im Attribut *Name* der Klasse *IfcRoot* vorhanden ist. Geeignet ist z.B. eine Kombination aus Klasse und laufender Nummer z.B. *IfcPipe_2458*.

Die IFC-STEP-Datei soll darüber hinaus folgende Voraussetzung erfüllen:

8. Kleine Dateigröße: Wissensgraphen sind i.d.R. verbosier als native Dateiformate (Abschnitt 3.8.4 zeigt dies beispielhaft für Modelica, Abschnitt 4.3.3 für den Vergleich zwischen IfcOWL und IFC-STEP). Unbedacht exportierte IFC-STEP-Dateien können schnell erhebliche Dateigrößen erreichen, die dann ggf. von IFC-Tools nicht mehr gehandhabt werden können (Tab. 6 führt dazu ein paar Erfahrungswerte auf). Werden diese dann in KG umgewandelt, steigen die Dateigrößen nochmals, ggf. auch auf ein Maß, welches weder von Texteditoren (Erfahrungswerte dazu finden sich in Tab. 58), noch von Ontologie-Tools (siehe Abschnitt 2.3.4) gehandhabt werden können.

Tab. 32 fasst die Anforderungen zusammen.

Tab. 32 Anforderungen an IFC-Files, die mit MO-x-IFC bearbeitet werden sollen

Nr	Voraussetzung	Verbindlichkeit	Verweis auf theoretische Grundlagen	Verweis auf Handlungsanweisung
1	Klassenzuordnung	MUSS	2.1.5	4.3.1.2
2	Typisierung	MUSS	2.1.5	4.3.1.2
3	Verbindungen	MUSS	2.1.9.1	4.3.1.3
4	Portbezeichnung	MUSS	-	-
5	Geschosszuordnung	SOLL	2.1.9.2	4.3.1.5
6	Systemzuordnung	SOLL	2.1.9.3	4.3.1.6
7	Bezeichnungen der IfcProducts	SOLL	-	4.3.1.7
8	Dateigröße	SOLL	2.1.2	4.3.1.8

Diese Voraussetzungen sind bei korrekter Anwendung mit Standardsoftware erfüllbar, wie das Beispiel in Abschnitt 5.1 zeigt. In Abschnitt 4.3.1 ist beschrieben, wie nicht-anforderungskonforme Modelle ohne Autorentools nachbearbeitet werden können.

3.5.2 Anforderungen aus dem Handling mit Modelica-Tools

Es ergibt sich jedoch nicht nur aus Gründen der Rechenzeit eine Anforderung an die Größe des Funktionsmodells, sondern auch aufgrund der notwendigen Benutzerinteraktion: Für das Handling der Simulationsmodelle in einer Benutzeroberfläche ist es entscheidend, die Anzahl der darzustellenden Elemente zu begrenzen. Modelica unterstützt dies durch die Möglichkeit der Verschachtelung: Dabei werden „Submodelle“ gebildet, welche – mit einem eigenen Icon versehen – als Komponenten im „Supermodell“ dargestellt werden. Eine starke Verschachtelung wirkt sich jedoch negativ auf die Übersichtlichkeit der Modelle aus, sie sollte daher vorrangig genutzt werden, um verschiedene Level of Detail zu unterscheiden.

Es wird angestrebt eine Darstellbarkeit der Simulationsmodelle (ohne Nutzung von Verschachtelung) auf einer Bildschirmseite zu erreichen bei gleichzeitiger Lesbarkeit der Texte. Abb. 68 (S. 165) zeigt ein solches Modell, welches knapp die Anforderungen an Übersichtlichkeit und Größe der Darstellung erfüllt. Dargestellt ist der Diagram-Layer eines Modells mit der Abmessung BxH 1400x1000 Größeneinheiten (GE). Ohne Anwendung eines speziellen Zoomfaktor entspricht eine solche GE in etwa einem Bildschirmpixel. Dementsprechend sind auf üblichen

Desktopmonitoren Querformate von ca. 2000x1000 GE gut darstellbar. Die Icons der Standardkomponenten incl. der zugehörigen Beschriftung sind i.d.R. so gestaltet, dass sie bei einer Darstellung in Standardgröße (20x20 GE, entspricht einem Kästchen in Abb. 68) gut erkennbar sind. Für eine übersichtliche Platzierung insb. auch der Verbindungslien zwischen den Komponenten sind ausreichende Abstände und eine Platzierung auf dem Raster notwendig. In Abb. 68 wurde als Abstand auch jeweils 20 GE in beide Richtungen gewählt. Es ergibt sich, dass ca. 50 Komponenten-„Spalten“ nebeneinander, sowie 25 Komponenten-„Zeilen“ (im Beispiel vorhanden: 29) übereinander platziert werden können. Im Beispiel sind 200 Komponenten enthalten. Dies ist eine erstrebenswerte Größenordnung für die maximale Größe automatisch erzeugter Modelle.

Das Platzangebot auf dem Modelica-Diagram-Layer entspricht in etwa dem Platzangebot eines A2-formatigen Papierplans und ist damit ausreichend für viele Anlagenschemata. Für komplexere Anlagen, muss auf Verschachtelung zurückgegriffen werden.

3.5.3 Vereinfachungsalgorithmus

Aus der Diskrepanz zwischen Anzahl der Komponenten im Konstruktionsmodell und Platzangebot im Diagram-Layer ergibt sich die Notwendigkeit, die Anzahl der Komponenten zu reduzieren, dabei sollen keine simulationsrelevanten Informationen verloren gehen. Das Konstruktionsmodell wird dabei als Graph betrachtet. Alle enthaltenen IfcProduct (siehe Abschnitt 2.1.4.3) werden als Knoten betrachtet. Die *IfcRelConnectsPorts*-Relationen (siehe Abschnitt 2.1.9.1) bilden die Kanten.

Zunächst werden alle *IfcProducts* im Konstruktionsmodell entsprechend ihres Knotengrades klassifiziert:

- *Loner*: Als „loner“ werden alle IfcProduct ohne jegliche Verbindung bezeichnet. Dies sollte in anforderungskonformen (s.o.) Modellen nicht vorkommen und ist daher ein Hinweis auf ein fehlerhaftes/nachzuarbeitendes Modell
- *Leaf*: Als „leaf“ werden alle IfcProduct mit nur einer Verbindung zu anderen Komponenten bezeichnet, praktisch trifft dies z.B. auf Druckhaltungen oder Druckmessstellen (siehe z.B. Abb. 63) zu.
- *Pipe*: In der Klasse „pipe“ werden alle Elemente mit genau zwei Verbindungen zu anderen Elementen zusammengefasst. In dieser Klasse sind mit Abstand die meisten Elemente einzurordnen.
- *Hub*: Als „hub“ werden alle Elemente mit mehr als zwei Verbindungen zu anderen Komponenten bezeichnet, typische Vertreter sind T-Stücke, 3-Wegeventile, Wärmeübertrager.

In Reihe geschaltete *pipe*-Elemente, welche keine besondere Funktion erfüllen, jedoch einen Druckverlust im System repräsentieren, können zusammengefasst werden, um den Netzwerkgraph zu vereinfachen. Da Druckverluste gerade bei Untersuchungen von hydraulischen Fehlerfällen entscheidend sind, werden derartige nicht-strukturrelevante Bauteile zu *replacements* zusammengefasst, jedoch nicht gelöscht. Die Zusammenfassung der nicht-strukturrelevanten *pipe*-Elemente zu *replacements* erfolgt entsprechend ihrer Klassifizierung und Typisierung, daher ist es eine Grundvoraussetzung, dass diese im Modell korrekt sind. In Tab. 31 sind die nicht-strukturrelevanten Bauteile markiert: dazu gehören z.B. gerade Rohrstücke, Bögen und Flansche.

Insbesondere hinsichtlich der Ventile ist die korrekte Typisierung (siehe Abschnitt 3.5.1 Nr. 3) hier wichtig. Alle automatisch durch die Gebäudeautomation betätigten Ventile müssen als *REGULATING* typisiert sein. Auch Rückschlagventile (*CHECK*) und 3-Wege-Ventile (*MIXING*) bleiben in der Simulation erhalten.

Es wurde ein Ersetzungsalgorithmus unter Nutzung der Python-Bibliotheken `ifcopenshell` und `networkx` (siehe Tab. 54) entwickelt. Dieser ersetzt Reihenschaltungen der o. g. Elemente durch ein *replacement* solange bis der Algorithmus auf ein strukturrelevantes Element stößt. Strukturrelevant sind dabei alle Elemente mit mehr als zwei oder nur einem Port, sowie Elemente mit zwei Ports, die gemäß der Definition auf S. 31 als strukturrelevant klassifiziert sind, Tab. 31 enthält dazu eine Übersicht. Weiterhin ist sichergestellt, dass keine Ersatzelemente für einzelne Elemente erstellt werden, da dies nur zusätzlichen Datenoverhead erzeugen würde.

3.5.4 2D-Platzierung

Wie bereits thematisiert wurde, werden die Anlagenstrukturen als Graphen interpretiert. Dabei stellt jedes Bauteil einen Knoten dar. Verbundene Bauteile werden durch eine Kante im Graph repräsentiert. Der Name der Datenstruktur „Graph“ impliziert bereits, dass diese besonders gut erfasst werden kann, wenn sie dargestellt wird. Üblich ist dabei die Repräsentation der Knoten als Punkte und der Kanten als Verbindungslien dieser Punkte. Auch wenn mathematisch gesehen andere Arten der „Abbildung“ denkbar sind, beschränken sich die folgenden Ausführungen auf Abbildungen in ein zweidimensionales Koordinatensystem (Papier, Bildschirm).

3.5.4.1 Anforderungen

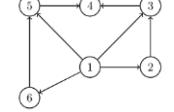
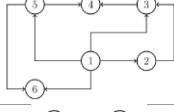
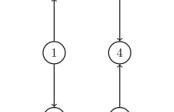
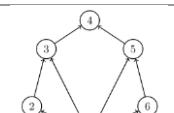
Das Ziel der Darstellung ist ein Erkenntnisgewinn/Verständnis für den Betrachter, daher sind Sehgewohnheiten der Zielgruppe unbedingt zu berücksichtigen. Abb. 50 zeigt ein typisches Schaltschema einer Anlage. Folgende ästhetische Merkmale sind für derartige Darstellungen charakteristisch:

1. Rechtwinkliger Verlauf der „Kanten“/Linien
2. Möglichst Kreuzungsfreie Linienführung, erreicht durch Anordnung in Strömungsreihenfolge („Abwicklung“ des Rohrnetzes)
3. Berücksichtigung der Etagenstruktur
4. Benutzung von Symbolen statt einfacher Punkte für die Knoten
5. Beschriftung von Kanten und Knoten
6. Farbliche Codierung der Kanten

3.5.4.2 Grundformen von Graph-Darstellungen

Tab. 33 zeigt Grundformen von Graphendarstellungen. Anlagenschemata können demnach als orthogonale Graphen bezeichnet werden. Ein entsprechender Layout-Algorithmus ist gemäß Tab. 34 zu wählen. Diese Algorithmen dienen der Ermittlung der Position der Knoten im Darstellungsbereich, sowie der Festlegung der Kantenführung.

Tab. 33 Graphen nach (Baeldung 2020)

Bezeichnung	Definition, Vorteile, Begrenzungen	Beispiel*
Geradliniger Graph	Alle Kanten sind Geraden. Diese Art von Diagramm vereinfacht die Identifizierung von Zyklen zwischen Knotenpunkten, da sie sich in unmittelbarer Nachbarschaft befinden.	
Orthogonaler Graph	Alle Linien, Kanten oder Teile davon müssen orthogonal oder parallel zueinander sein. Orthogonale Graphen heben den Knotengrad eines jeden Knotenpunkts hervor, weil es sehr einfach ist, die Anzahl der einfallenden Kanten zu ihnen zu zählen. Begrenzung: alle Knotengrade <= 4 (im 2D), <=6 (im 3D)	
Orthogonaler geradliniger Graph	Graphen, die gleichzeitig die Orthogonalität erfüllen und deren Kanten Geraden sind. Diese Art Graph hebt die Bedeutung der Graphflächen hervor, die den endlichen Flächen entsprechen, die von den Kanten des Graphen begrenzt werden, einschließlich der unendlichen Fläche, die sie umgibt. Begrenzung: alle Knotengrade <= 4 (im 2D), <=6 (im 3D)	
Gerichteter Graph	Kanten haben eine Richtung	

*Bildquelle: (Baeldung 2020)

3.5.4.3 Gerichtete und ungerichtete Graphen

Zu diskutieren ist hierbei noch der Punkt der Flussrichtung: Anlagenschemata enthalten i.d.R. keine Darstellung der Flussrichtung, da diese implizit enthalten ist: *Vorlaufleitungen* beschreiben die Flussrichtung vom Erzeuger zum Verbraucher und *Rücklaufleitungen* die Flussrichtung vom Verbraucher zum Erzeuger. Verbindungen zwischen Vor- und Rücklauf kommen nur ausnahmsweise in folgenden Fällen vor:

- A. Beimischschaltungen zur Annäherung beider Temperaturen, konkret in einem Heizsystem
 - a. Vorlauftemperaturabsenkung z.B. für Fußbodenheizkreise
 - b. Rücklauftemperaturanhebung z.B. für BHKWs
- B. Am Erzeuger
- C. Am Verbraucher
- D. Überströmverbindungen zum Druckausgleich

In allen Fällen befindet sich an der Verbindungsstelle zwischen Vor- und Rücklauf ein Bauteil/Armatur. Mit Ausnahme der Überströmverbindungen ist die intendierte Strömungsrichtung immer klar, wird dennoch im Schema nicht durch Pfeile gekennzeichnet.

Dies ist insofern sinnvoll, als Rohre wie auch die meisten anderen Bauteile, rein technisch keine Festlegung hinsichtlich der Strömungsrichtung haben (Ausnahme: Rückschlagklappe, 3-Wege-Ventile). Die in Abschnitt 2.2.1 beschriebene akusale Modellierung in Modelica trägt diesem Umstand Rechnung und ermöglicht es insofern neben dem Regelbetrieb auch hydraulische Fehlerfälle (nichtintendierte Strömungsmengen und -richtungen) abzubilden. Dementsprechend sind weder für die Darstellung im Schema, noch für die Überführung nach Modelica Angaben zur Strömungsrichtung nötig. Für den Layout-Algorithmus des Graphen und die Festlegung der „Positiven“ Strömungsrichtung können sie eine hilfreiche Metainformation sein. In vielen IFC-Dateien ist diese Information jedoch nicht enthalten, weshalb auf ihre Nutzung verzichtet wird.

3.5.4.4 Layout-Algorithmen

Es gibt zahlreiche wissenschaftliche Veröffentlichungen zu Algorithmen für das Layouting von Graphen, viele davon stehen als FLOSS-Implementierungen z.B. innerhalb von Tools wie Gephi zur Verfügung. Diese Layout-Algorithmen können wie folgt klassifiziert werden:

- Labelbasierte Anordnung: es erfolgt eine Anordnung in alphabetischer Reihenfolge z.B. auf einer Achse oder einem Kreis. Einzelne Knoten können anhand ihres Labels gut aufgefunden werden.
- Meta-Daten-basierte Algorithmen: die Anordnung erfolgt anhand von Meta-Daten der Knoten. Dies setzt das vorhandensein von Meta-Daten an den Knoten voraus. Für die hier betrachteten Anlagengraphen, welche als IFC-STEP-Dateien vorliegen, können die *Attribute* und *Properties* der *IfcProduct* als solche Meta-Daten interpretiert werden.
- Kraftbasierte Algorithmen: Die Kanten des Graphen werden als Federn betrachtet und die Knoten als sich abstoßende Massen. Es wird eine Anordnung gewählt, welche sich bei Kräftegleichgewicht einstellt. Dies sorgt für eine Darstellung mit minimalen Kantenüberlappungen.

Die Layout-Algorithmen unterscheiden sich nach Eignung für verschiedene Graphgrößen/-dichten, hinsichtlich der Laufzeit und des produzierten Ergebnisses. Viele davon legen folgende Layout-Grundprinzipien zugrunde, die eine Darstellung für einen menschlichen Nutzer „übersichtlich“ erscheinen lassen:

- Minimierung Kantenkreuzungen
- Minimaler Flächenverbrauch der Graphen
- Minimierung Knicke (Richtungsänderungen der Verbindungslien)
- Maximale Größe des minimalen Winkels
- Maximale Symmetrie

Ein wesentliches Kriterium bei der Auswahl eines Layoutingalgoritmus ist die Größe und Vernetzungscharakteristik des Graphen: Die hier thematisierten Graphen repräsentieren Rohrnetze und sind relativ klein (Knotenzahl vierstellig) und *dünn*, d.h. nicht besonders eng vernetzt: i.d.R. wird jeder Knoten zwei Verbindungen zu seinen Nachbarknoten haben.

Tab. 34 open-source Layout-Algorithmen für Graphen

Bezeichnung	Verfügbarkeit **	Charakteristik	Größe des Graphen (Anzahl Knoten)	Quelle (paper)
neato*	Networkx Graphviz	kraftbasiert	< 100	(Kamada und Kawai 1989)
spring	Networkx Gephi	Kraftbasiert, nutzt kein Kantengewicht	< 1000	(Fruchterman und Reingold 1991)
sfdp	Graphviz Gephi	Kraftbasiert, nutzt kein Kantengewicht	100...100.000	(Hu 2005)
OpenOrd	Gephi	Kraftbasiert, nutzt Kantengewicht	100...1.000.000	(Martin u. a. 2011)
circular shell bipartite multipartite spiral	Networkx	Labelbasiert / Meta-Daten-basiert		
planar	Networkx	Minimiert Kantenkreuzungen		
ForceAtlas(2)	Gephi	Kraftbasiert, Nutzt Kantengewicht	1...1.000.000	(Jacomy u. a. 2014)
dot	Graphviz	Nutzung der Knotenreihenfolge, Minimierung von Kantenkreuzungen, Geschichtete Anordnung Richtungsinformation wird aber genutzt		

* In dieser Arbeit genutzt

** zu den Tools siehe Tab. 54

Abb. 66 zeigt die Darstellung des gemäß Abschnitt 3.5.3 vereinfachten Graphen mit dem kraftbasierten Algorithmus *neato*. Dabei werden bereits die Anforderung 2 und 4 gemäß der Aufzählung auf S. 104 erfüllt. Damit ist es möglich als menschlicher Bearbeiter mit diesem Modell weiterzuarbeiten.

3.6 MoOnt Modelica-Ontologie²⁹

Um Modelica-Modelle in Wissensgraphen überführen zu können ist ein Vokabular nötig, welches alle für die UseCases wichtigen Informationen beschreiben kann - dazu gehören insb. Schnittstellen und Parameter von Modelica-Modellen (sog. *shape*). Die in Abschnitt 1.2.3.3 beschriebene Recherche zu vorhandenen Ontologien in der Domäne Anlagen- und Regelungstechnik hat ergeben, dass diese nicht ausreichend sind, um Modelica-Modelle abzubilden. Es wurde daher eine Ontologie für die Sprachkonstrukte namens *Modelica Ontology* (Kürzel: *MoOnt*) entwickelt

Wie in Abschnitt 2.2 bereits beschrieben wurde unterscheidet Modelica auf der Sprachebene nicht zwischen Bibliotheks-Komponenten und *Modelica-Executables*. Dementsprechend wird dieses Paradigma auch für *MoOnt* vorgesehen. Die Ontologie ist gleichermaßen für die Beschreibung von Bibliotheks-Komponenten und Modelica-Executables vorgesehen.

Für eine Entwicklung von Ontologien gemäß der Vorgehensweise des Knowledge Engineering (Pinto und Martins 2004; Schneider u. a. 2020) sind zunächst die Anforderungen zu spezifizieren. Diese ergeben sich aus den Anwendungsfällen, die in Abschnitt 1.5.1 beschrieben wurden, und können wie folgt zusammengefasst werden.

- Abbildung der *shape* eines Modells, d.h. Konnektoren und Parameter
- Abbildung von Verbindungen zwischen Komponenten mit Hilfe ihrer Konnektoren
- Abbildung der Klasse von Komponenten insb. von Konnektoren und Parametern

Als nächster Schritt des Knowledge Engineering folgt „Conceptualisation“, dabei werden die notwendigen Begrifflichkeiten und ihre Beziehung zueinander zusammen getragen. In dem Zuge sollte auch geprüft werden, inwiefern bereits Ontologien existieren, welche weiter genutzt werden können. Die *Wolfram System Modeler (wsm)* Ontologie (Wolfram Research 2014) wurde bereits in Abschnitt 1.4.3 thematisiert. Sie wurde entwickelt um Modelica-Modelle abzubilden, jedoch nach ihrer Veröffentlichung im Jahr 2014 nicht weiter gepflegt. Sie wurde daher als Basis für MoOnt verwendet, indem ihre Inhalte kopiert (nicht referenziert) wurden. Abb. 36 zeigt das zugehörige Klassendiagramm. Markiert sind darin nach *MoOnt* übernommene und nicht übernommene Konzepte, sowie Konzepte, die zwar im Klassendiagramm enthalten sind, jedoch im zugehörigen Ontologie-File nicht.

²⁹ Dieser Abschnitt wurde von der Autorin in ähnlicher Form bereits veröffentlicht: (Eckstädt, Menzel, u. a. 2023)

3 Methodischer Ansatz

3.6 MoOnt Modelica-Ontologie

3.5.4 2D-Platzierung

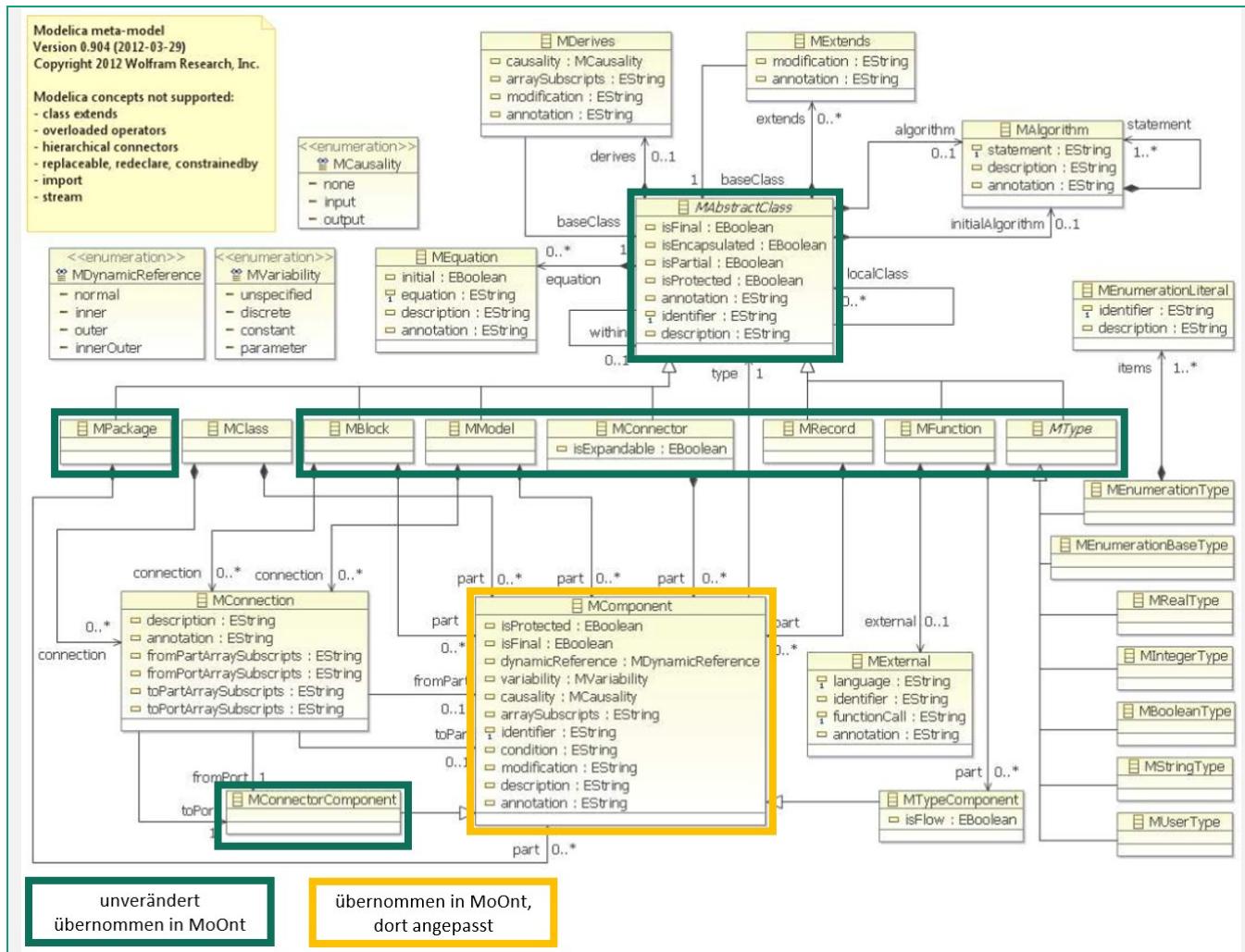


Abb. 36 Klassendiagramm von wsm, Bildquelle: angepasst Darstellung basierend auf (Wolfram Research 2014)

Abb. 37 zeigt das Ergebnis der Conceptualisation als Klassen-Diagramm, welche im Folgenden näher beschrieben werden:

MAbstractClass ist eine abstrakte Sammelklasse für die in Modelica Sprachstandard vorhandenen Klassen, welche jeweils mit einer Klasse auch in MoOnt abgebildet wurden³⁰: **model** und **package** sind davon die am häufigsten genutzten. Instanzen der Klassen **MModel**, **MPackage** und **MConnector** können als Komponenten anderer Klassen erzeugt werden. Da Konnektoren und Parameter für die Übertragung aus IFC-Modellen eine herausgehobene Rolle spielen (siehe Anforderungsliste oben), wurden dafür separate Entitäten (**MConnectorComponent**, **MParameterComponent**) definiert, letztere wurden gegenüber wsm ergänzt.

MComponent wird mit der Relation **type** eine Klasse zugeordnet. Sie haben als Attribute eine Bezeichnung **identifier**, ggf. auch eine Kommentar **stringComment** (entspricht „documentation string“ aus der MLS) und eine Modifikation **modification**. Diese kann z.B. dazu dienen einen

³⁰ Für Entitäten, welche im Modelica-Sprachstandard vorkommen, wurde der Präfix „M“ gewählt, um Verwechslungen mit allgemeinen Schlüsselwörtern wie „Klasse“ auszuschließen.

Parameter zu belegen. Diese „Attribute“ in der Darstellung Abb. 37 wurden als *DataTypeProperties* in der Ontologie vorgesehen.

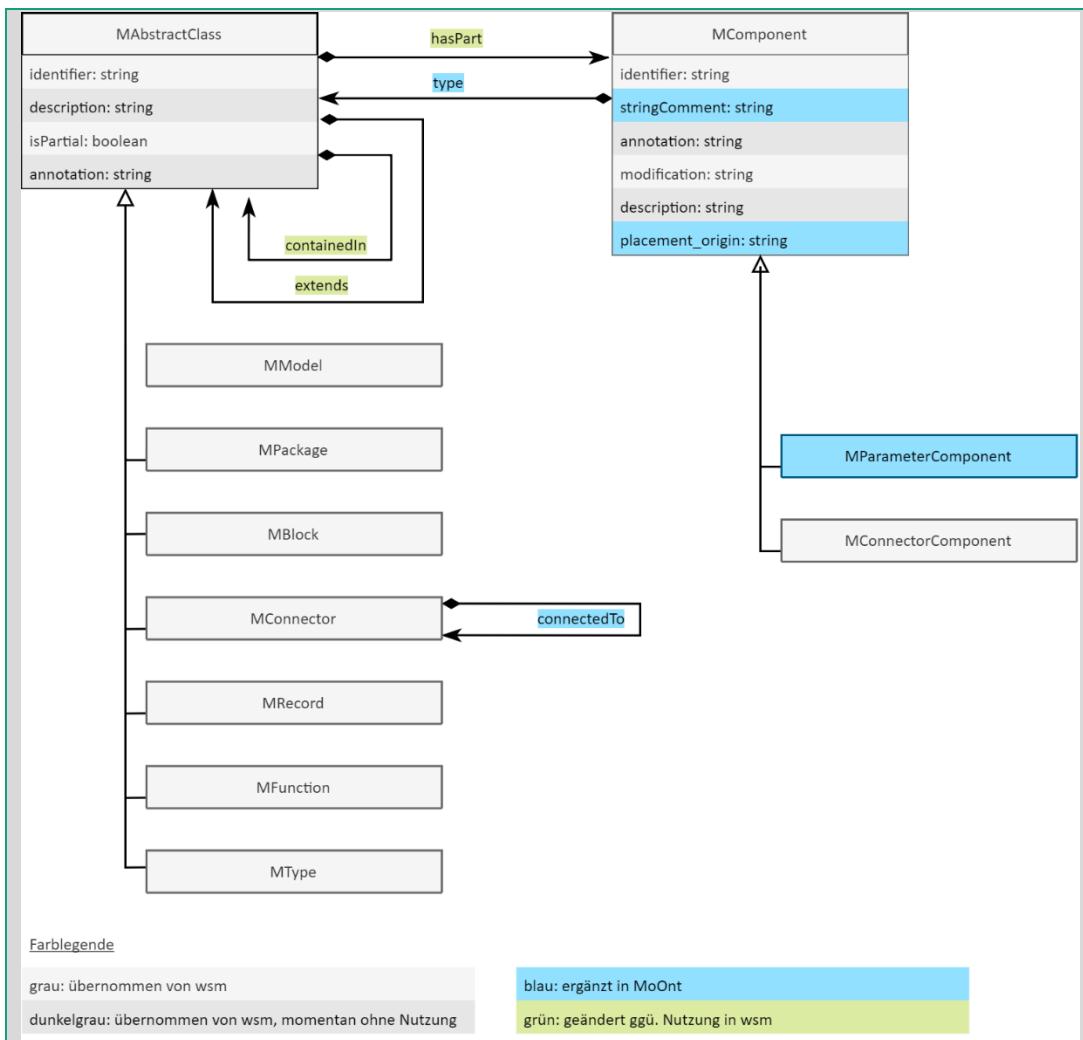


Abb. 37 UML Diagramm MoOnt

Die Relationen `containedIn` und `extends` beschreiben die Package- und Vererbungsstruktur von Modellen und sind insb. für die Analyse von Simulationsbibliotheken hilfreich. Sie wurden in *MoOnt* gegenüber *wsm* ergänzt. *MAbstractClass* ist dabei sowohl Definitions- als auch Wertebereich (*domain* und *range*). Die Abbildung der Package- und Vererbungsstruktur war in *wsm* über eigene Entitäten *MExtends* und *MDerives* vorgesehen, für den in dieser Arbeit vorgesehnen Anwendungsfall ist diese zusätzliche Komplexität allerdings hinderlich, weshalb die Abbildung mit Relationen vorgesehen wurde.

Die Implementierung der Ontologie erfolgte mit dem frei verfügbaren Ontologie-Editor Protégé (siehe Abschnitt 2.3.4.2). Sie enthält 267 Axiome und wurde als ttl-File exportiert. Sie steht online³¹ zur Verfügung. Es wurde der *namespace prefix* *moont* bei w3id.org³² dafür registriert. Eine erste Version von *MoOnt* wurde 2023 veröffentlicht (Eckstädt, Menzel, u. a. 2023).

³¹ https://github.com/ElisEck/MO-x-IFC/tree/main/TBox/ontologies/7_MoOnt

³² w3id.org/moont

3.7 Library-Ontologien

Modelica ist eine objekt-orientierte Modellierungssprache. Demnach ist es erstrebenswert und üblich, dass Modelica-Modelle aus Bibliotheks-Komponenten aufgebaut werden. Um die Aussagekraft des zu einem spezifischen Simulationsmodell gehörigen KG zu erhöhen, ist es daher sinnvoll (und für bestimmte Fragestellungen auch nötig), auch die Informationen über die enthaltenen Bibliotheks-Komponenten in den KG zu übertragen.

Basierend auf MoOnt wurden daher Ontologien für die Simulationsbibliotheken wie z.B. die *ModelicaStandardLibrary* und die *ModelicaBuildingsLibrary*, sowie die *AixLib* entwickelt. Wie in Abb. 31 dargestellt bauen diese Bibliotheken hierarchisch aufeinander auf. Die Libraries wurden in Abschnitt 2.2.6 beschrieben.

Es wurde für die Erstellung der KG zu den Simulationsbibliotheken eine analoge Vorgehensweise gewählt wie sie für die IFC-Ontologie *IFCowl* praktiziert wurde. Auch dort wurde unterschieden in eine Ontologie für die Formulierungssprache (bei IFC: EXPRESS) und eine Ontologie für das Datenschema an sich - Tab. 35 zeigt diese Analogie.

Tab. 35 Gegenüberstellung Ontologien im Bereich IFC und Modelica

	IFC	Modelica		
Datenschema	IFC2x3	IFC4.0.2.1	MSL	MBL
... zugehörige Ontologie	standards.buildingsmart.org/IFC/DEV/IFC2x3/FINAL/OWL	standards.buildingsmart.org/IFC/DEV/IFC4/ADD2_TC1/OWL	w3id.org/msl	w3id.org/mbl
Formulierungssprache	EXPRESS			Modelica Language Specification
... zugehörige Ontologie IRI	w3id.org/express			w3id.org/moont

Abb. 38 zeigt, wie die verschiedenen Modelica-Bibliotheken aufeinander aufbauen (rechte Seite der Abbildung). Parallel zu den Schichten auf der Modelica-Seite gibt es passende Ontologien (linke Seite), die verwendet werden, um die Modelle dieser Ebene semantisch darzustellen. Unterhalb der im Folgenden beschriebenen Library-Ontologien wird die MoOnt verwendet, die ihrerseits auf den spezifizierten W3C-standardisierten Ontologien basiert (diese unteren Schichten sind detaillierter dargestellt in Abb. 31). Die MoOnt repräsentiert die Modelica Language Specification, welche jedoch keine Modelica-Bibliothek ist. Ein Pendant zu den Basisontologien existiert auf Seite der Modelica-Libraries nicht – genau dies verursacht den Unterschied zwischen dem *Wissen* in der linken Pyramide und den *Informationen* in der rechten Pyramide (siehe Abschnitt 2.3.1).

Die Erzeugung der Library-Ontologien erfolgte automatisiert durch Nutzung des MoTTL-Transcriptor, der in Abschnitt 3.10 beschrieben wird.

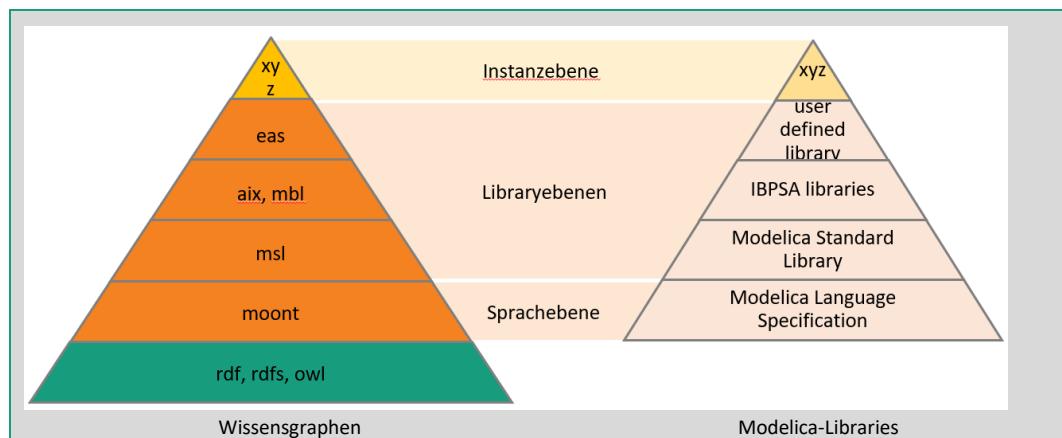


Abb. 38 Wissensgraphen-Stapel und zugehöriger Stapel von Modelica Bibliotheken

Tab. 36 fasst die wesentlichen Kennzahlen zu den entstandenen Library-Ontologien zusammen. Man erkennt, dass die automatisch generierten Ontologien (d.h. jene für die Modelica-Bibliotheken) deutlich umfangreicher sind, als die manuell erzeugte MoOnt. Man erkennt weiterhin, dass die Dateigröße gegenüber den nativen Files deutlich vergrößert ist. Annotationen (einschließlich der oft umfangreichen Dokumentation) wurden, mit Ausnahme der Information über die Platzierung der Icons der Komponenten, nicht in den KG übernommen. Gleichungen und Algorithmen wurden, abgesehen von den *connect-equations*, ebensowenig berücksichtigt. Dennoch ist die Dateigröße größer als die der vollständigen Bibliothek im nativen Format.

Obwohl die semantischen Repräsentationen aller Bibliotheken nur Informationen über die Vererbungshierarchie, die Paketstruktur und die Komponenten der Modelle enthalten, ist die Dateigröße der semantischen Repräsentation groß. Dies erkennt man auch in der Gegenüberstellung der Dateiausschnitte in Lis. 15 und Lis. 16: Die Darstellung als Wissengraph ist deutlich länger, obwohl sie nur einen Teil der Informationen enthält. Das Modelica *package*, welches das *ModelicaExecutable* aus Lis. 16 enthält, benötigt 30kB Festplattenspeicher, während der Wissensgraph (in Turtle-Serialisierung) 88 kB belegt – fast die dreifache Größe. Der KG enthält 1063 Axiome, während das native Modelica File nur 600 Textzeilen umfasst. Diese Dateivergrößerung ist eine grundsätzliche Begleiterscheinung bei der Umwandlung nativer Formate in Wissensgraphen. Problematisch große Dateien entstehen dabei allerdings vor allem auf Seite der IFC-Dateien – Modelica-Dateien sind i.d.R. so klein, dass auch die vergrößerten Dateien noch gut handhabbar sind. Problematische Größenordnungen können Tab. 58 im Anhang entnommen werden.

Tab. 36 Kennzahlen der entwickelten Ontologien

Ontology prefix	Axiome	Klassen	Individuals	Filesize (ttl serialization)	Filesize (Modelica-Files)
MoOnt	267	30	3	23,4 kB	-
MSL	186.228	6.706	51.147	20,8 MB	14,0 MB
Aix	432.751	11.091	106.757	24,9 MB	14,5 MB
MBL	365.890	5.600	60.064	42,2 MB	11,6 MB

Die *Modelica Standard Library* MSL ist die Basisbibliothek für alle Modelica-Modelle. Genau wie die Modelica-Sprachbeschreibung wird sie von der Modelica Association gepflegt. Sie enthält domänenunabhängige Basismodelle, z.B. allgemeine Anschlussdefinitionen wie *HeatPort*, *FluidPort*, *RealPort*, Einheitenumwandler, Mediendefinitionen. Tab. 36 zeigt dass die Ontologie mit 6.700 Klassen und 51.000 *Individuals* immens groß ist.

Die *Modelica Buildings Library* MBL besteht im Wesentlichen aus 12 Paketen und wird in Form von mehr als 4500 Modelica-Files (Textdateien) bereit gestellt. Jede Datei enthält i.d.R. eine, ausnahmsweise auch mehrere, Modelica-Klassen. Insgesamt benötigen sie 12 MB Speicherplatz. Der transkribierte KG umfasst 366.000 Axiome und das entsprechende turtle-File benötigt 42 MB Speicherplatz. Die Dateistruktur der *AixLib* aix ist ähnlich, der zugehörige KG ist daher auch in derselben Größenordnung.

Alle Library-Ontologien sind nicht für den manuellen Gebrauch gedacht, sondern vor allem für Schlussfolgerungen im Zusammenhang mit den Instanz-Graphen wichtig. Die Größe der Ontologien verursacht Anforderungen an die Hardware der verarbeitenden Rechner, insb. geht damit Bedarf an Arbeitsspeicher einher.

3.8 SemTran Semantic Translator (Konverter 4)

3.8.1 Allgemeines

Abb. 35 zeigt das Zusammenspiel der verschiedenen Wissensgraphen bei der *Semantic Translation*.

Auf der Instanzebene (A-Box) sind die verschiedenen Modelle gemäß Tab. 29 dargestellt. Diese werden zunächst mittels der Transkriptions-MOXIK in einen Wissensgraph überführt.

Die dafür notwendigen Vokabulare befinden sich auf der Schemaebene und sind vom konkreten Beispiel unabhängig, sie werden also nur einmalig erstellt. Sie treten ebenfalls in Form von KG, d.h. als ttl-Dateien auf. Das Vokabular ist jeweils mehrstufig definiert, aufbauend auf der Formulierungssprache (MLS oder EXPRESS) werden die konkreten Datenschemata spezifiziert. IFC besteht – wie im Abschnitt 2.1.2 beschrieben – aus verschiedenen Unter-Daten-Schemata, buildingsSMART hat alle diese Unter-Daten-Schemata in einer gemeinsamen Ontologie veröffentlicht. Die Modelica-seitigen Datenschemata sind die verschiedenen Modelica-Bibliotheken. Diese bauen hierarchisch aufeinander auf. Um eine Konsistenz zwischen den Modelica-Bibliotheken und den zugehörigen KG zu gewährleisten wurde entschieden, für jede Modelica-Bibliotheken eine eigene Ontologie mit eigenem Namespace zu erstellen. Die im Kontext dieser Arbeit wesentlichen Modelica-Bibliotheken sind in Tab. 35 aufgeführt und wurden im Abschnitt 3.7 detailliert beschrieben.

Um die beiden Domänen (IFC und Modelica) zu verbinden wird weiterhin das Alignment als Zuordnung benötigt, dieses wird in Abschnitt 3.8.4 thematisiert.

3.8.2 Teilschritte

Die „Semantic Translation“ ist ihrerseits in zwei Prozessschritte zu unterteilen, welche in Abb. 39 dargestellt sind:

- 1) **Reasoning:** Zunächst erfolgt die Zuordnung der Klassen aus der jeweils anderen „Sprache“ zueinander. Dafür werden Reasoner (☞ 2.3.3.3) und das Modelica-IFC-Alignment genutzt. Die dabei entstehenden Tripel werden in einem intermediären KG `xyz_classes.ttl` abgelegt. Dieser erfüllt nicht die auf S. 94 genannte Anforderung an ZS, dass alle relevanten Informationen in dieser Datei enthalten sind und erscheint daher auch nicht als ZS in der Übersichtgrafik in Abb. 33.

Dieser Teilschritt ist unabhängig von der Übersetzungsrichtung, Implementierungsdetails dazu sind in Abschnitt 4.3.3 beschrieben. Der Teilschritt ist auch über die in dieser Arbeit hauptsächlich verwendeten Sprachen Modelica und IFC hinaus anwendbar. Notwendig ist dafür jeweils ein Alignment zwischen den beiden beteiligten Domänen, welches die Funktion eines Wörterbuchs erfüllt. Das Alignment kann nicht allgemein für „Modelica zu IFC“ formuliert werden, sondern muss sich auf die konkret verwendeten Simulationsbibliotheken beziehen, ggf. sogar auf einen konkreten Versionsstand von IFC und der jeweiligen Simulationsbibliothek.

- 2) **Enhance:** Anschließend werden die relevanten Eigenschaften ergänzt. Dazu gehören insb. die Identifikatoren aus und für die Originaldateien – *GlobalIds* im Falle von IFC und *identifier* im Falle von Modelica. Dieser Teilschritt muss für jede Übersetzungsrichtung separat implementiert werden und weist somit einen prozeduralen Input-Output-Charakter auf. Implementierungsdetails dazu sind in Abschnitt 4.3.4.3 enthalten.

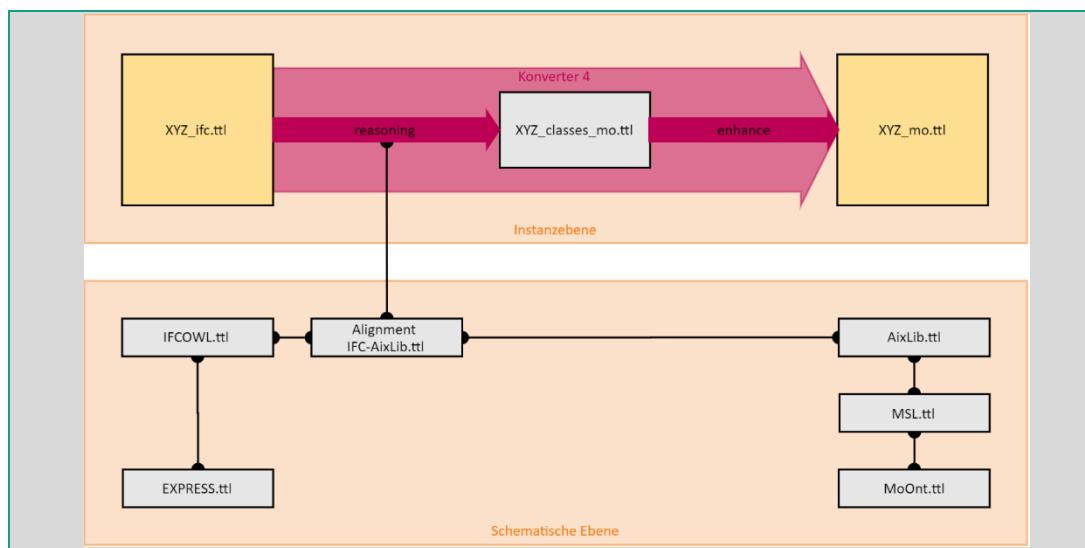


Abb. 39 Übersicht zum Translate Schritt zwischen Modelica und IFC

3.8.3 Vorteile des Semantic Reasoning

Die Nutzung von Reasonern für die Überführung von Wissensgraphen mit unterschiedlichem Vokabular hat folgende Vorteile im Vergleich zu einer Implementierung als „Procedural Translator“ (wie sie in Abschnitt 3.11 beschrieben wird):

- Reasoner sind bereits vorhanden und getestet. Es entsteht somit weniger Programmieraufwand und weniger Fehler auf der Implementierungsebene.
- im Falle einer bitotalen Zuordnung (siehe Abschnitt 3.8.4.2) ist die Richtung der Übersetzung flexibel und muss nicht separat programmiert werden
- Modularität: Trennung der Fachlichkeit (Erstellen des Alignments) von der Implementierung der Übersetzung (Programmierarbeit)
- Erweiterbarkeit, wenn Normung z.B. hinsichtlich IFC fortschreitet oder neue Modelica Library-Versionen entstehen
- Universelle Anwendbarkeit, solange ein Alignment vorhanden ist auch für die Übersetzung in andere Ontologien wie z.B. in Abb. 34 angedeutet.

Die Nachteile des Semantic Reasoning sind in Abschnitt 3.11 beschrieben.

3.8.4 IFC – Modelica – Alignment

3.8.4.1 Allgemeines

Alignments stellen die Verbindung zwischen verschiedenen Ontologien her. Die Schlüsselwörter dafür sind in den sog. Basis-Ontologien definiert und in Tab. 26 auf S. 89 aufgelistet. Alignments werden maschinenlesbar ebenfalls als Wissensgraphen (KG) abgespeichert. Für die Erläuterungen in diesem Abschnitt wird eine tabellarische Darstellung bevorzugt. Die entstandenen Alignment sind online als ttl-Files im MO-x-IFC-Repository³³ verfügbar.

Alignments sind eine Voraussetzung für das automatisierte Übersetzen durch Reasoner, sie können jedoch auch unabhängig davon z.B. für SPARQL-queries (siehe Abschnitt 2.3.2.6) über verschiedene KG hilfreich sein.

Ein Alignment zwischen IFC und Modelica kann nicht auf Sprachebene (EXPRESS, MLS) erfolgen, sondern nur in Bezug auf die einschlägigen Simulationsbibliotheken, wie sie in Abschnitt 2.2.6 beschrieben wurden. Für diese wurden mit Hilfe des MoTTL-Transcriptor KG-Modelle erzeugt (siehe Abschnitt 3.7).

Die Alignments in dieser Arbeit beschränken sich generell auf eine Zuordnung von Klassen zueinander, dafür wird die *owl:equivalentClass*-Relation genutzt. Dabei werden ggf. Nebenbedingungen entsprechend der in Abschnitt 3.8.4.2 beschriebenen Problemklasse berücksichtigt. Eigenschaften (*Attribute/Properties* in IFC, Parameter/Annotationen/Documentations-Strings in Modelica) werden nicht zugeordnet, da diese i.d.R. als *DataTypeProperty* im KG vorliegen und von Reasonern daher nicht ausgewertet werden können (W3C 2007; Demri und Quaas 2021; 2023; Lutz und Brandt 2007). Dies betrifft neben Eigenschaften im engeren Sinne insb. auch Meta-Eigenschaften wie z.B. die GlobalID und die Platzierung eines Elements. Diese werden im zweiten Teilschritt der Translation *enhance* (siehe Abb. 39) ergänzt.

Durch die Beschränkung des Alignments auf die *owl:equivalentClass*-Relation, entspricht das Alignment dem *OWL Lite* Profil (siehe Abschnitt 2.3.3.3). Die entstehende Reasoning-Aufgabenstellung ist daher einfach strukturiert und für Reasoner gut zu handhaben.

³³ https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/alignments/AIMAx_0.5.0.ttl

3.8.4.2 Problemklassen bei Zuordnungen

Bei Übersetzungen handelt es sich prinzipiell um Zuordnungen. In der Mengenlehre werden drei spezielle Arten von Zuordnungen zwischen Objekte aus zwei Mengen unterschieden, welche in Abb. 40 dargestellt sind:

- Linkseindeutig/inkettiv: jedem Element in Y ist maximal ein Element in X zugeordnet
- Rechtstotal/surjektiv: jedem Element in Y ist mindestens ein Element in X zugeordnet
- Bitotal/bijektiv: jedem Element in Y ist genau ein Element in X zugeordnet (inkettiv + surjektiv)

Als **eineindeutig** wird eine Zuordnung bezeichnet bei der jedem Element aus Y höchstens ein Partner aus X zugeordnet wird und umgekehrt jedem Element aus X höchstens ein Partner aus Y zugeordnet wird. Lücken in der Zuordnung sind damit nicht ausgeschlossen.

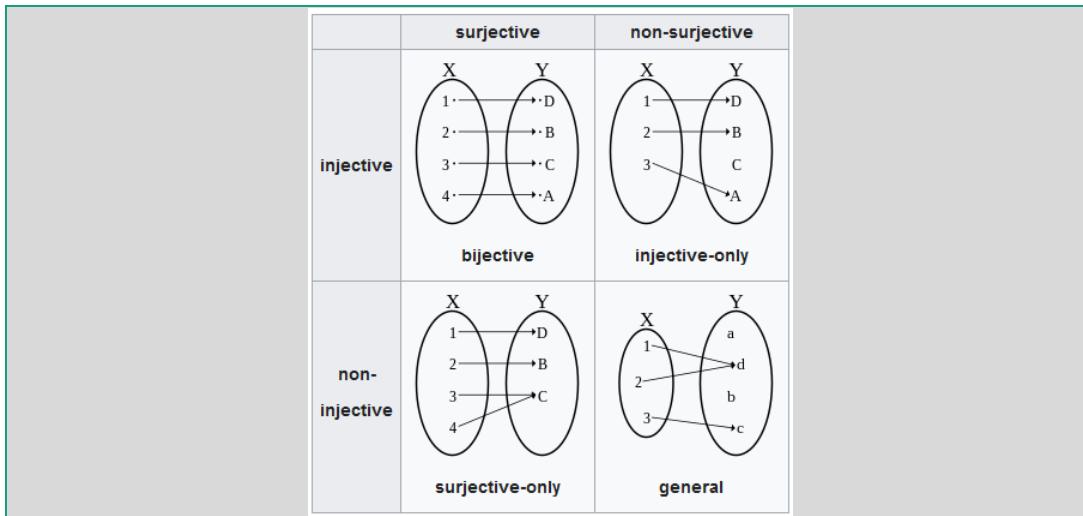


Abb. 40 Arten von Zuordnungen im Allgemeinen

Bildquelle: https://en.wikipedia.org/wiki/Bijection,_injection_and_surjection

Um die Vorteile des semantic reasoning nutzen zu können, d.h. die automatische Bidirektionale Übersetzung, ist es notwendig eine bitotale Zuordnung gem. Abb. 40 zwischen den relevanten Teilmengen von Modelica-Bibliotheks-Komponenten und IFC-Klassen³⁴ vorzunehmen. Dafür können 6 Schwierigkeitsstufen/Problemklassen identifiziert werden, die in Tab. 37 beschrieben werden.

Tab. 37 Problemklassen in Bezug auf Übersetzung

Nr	Beschreibung
0	Im einfachsten Fall bestünde eine eindeutige Zuordnung ohne weitere Bedingungen zwischen Klassen in IFC und Modelica. Dies ist für Zuordnungen zwischen Schemata verschiedener Domänen fast ausgeschlossen. <i>Beispiel: kein Beispiel im Kontext dieser Arbeit</i>
1	Die Zuordnung ist auf Klassenebene uneindeutig, es kann jedoch mit Hilfe von obligatorischen Attributen eine Eindeutigkeit hergestellt werden <i>Beispiel: Zuordnung von AixLib.Fluid.Movers.Flow:Controlled_m_flow. zu IfcFan oder IfcPump je nach enthaltenem Medium. (siehe Tab. 38 und Tab. 39)</i>
2	Das IFC-Schema spezifiziert nur sehr wenige Attribute als verpflichtend, jedoch gibt es Aufzählungen von <i>predefinedTypes</i> im Schema (z.B. <i>IfcPumpTypeEnum</i>), sowie <i>IfcPredefinedPropertySets</i> . Kann mit deren Hilfe eine Eindeutigkeit hergestellt werden, so wird diese Zuordnung in Problemklasse 3 eingesortiert, da es eine höhere Verbindlichkeit aufweist als Zuordnungen aus Klasse 4. <i>Beispiel: alle Zuordnungen in Tab. 38</i>
3	Wenn vordefinierte Typen oder PropertySets nicht ausreichen, muss die eindeutige Zuordnung mittels benutzerdefinierter Eigenschaften erfolgen. Technische Möglichkeiten dazu sind für IFC in den Abschnitten 2.1.7 und 2.1.9, für Modelica in Abschnitt 2.2.2.4 beschrieben. Um benutzerdefinierten Eigenschaften eine höhere Verbindlichkeit zuzuweisen, sollte auf die in Abschnitt 1.2.1 beschriebenen Prozesse zurückgegriffen werden. BIM-Abwicklungspläne (BAP) und Auftraggeber-Informations-Anforderungen (AIA) sind geeignete Dokumente, um die projektspezifische Nutzung von Eigenschaften insb. <i>IfcPropertySets</i> zu regeln. <i>Beispiel: Zuordnung von IfcFan zu Supply_Fan oder Exhaust_Fan in BRICK je nach Zugehörigkeit zum System Abluft und Zuluft mit IfcRelAssignsToGroup</i>
4	Wenn auch dies nicht möglich ist, kann die Zuordnung auch auf Basis der inhaltlichen Auswertung von Eigenschaften erfolgen <i>Beispiel: Unterscheidung zwischen Heating Coil or Cooling Coil basierend auf den angegebenen Auslegungs-Temperaturen</i>
5	Als letzte Option wird die Nutzung von Namenskonventionen betrachtet.

Alternativ zu Namenskonventionen bzw. allen in den Problemklassen 1-5 beschriebenen Hilfskonstrukten, kann auch einer Erweiterung der Ontologie in Betracht gezogen werden. Diese Option wird in Abschnitt 3.8.4.4 beschrieben, sie widerspricht jedoch der Anforderung „Nutzung standardisierter Formate“, die auf S. 94 formuliert wurde. Wenn eine Entsprechung auf einer der beiden Seiten völlig fehlt, dann muss eine Erweiterung der Ontologie vorgenommen werden. Während dies für Modelica einfach möglich ist durch eigenes Erstellen von Komponenten, ist dies bei IFC deutlich schwerer, da hierüber die pflegenden Organisationen (siehe Abschnitt 2.1.1) entscheiden.

3.8.4.3 IFC-AixLib für hydraulische Komponenten

Ein Alignment wurde in Hinblick auf die AixLib formuliert, wäre jedoch in sehr ähnlicher Form auch für die ModelicaBuildingsLibrary (MBL) gültig, da insb. die für das Alignment relevanten Komponenten aus dem gemeinsamen Kern (dem sog. IBPSA-core) stammen.

Tab. 38 fasst die Zuordnungen zwischen IFC und AixLib für hydraulische Komponenten zusammen, diese wurde entsprechend dem avisierten Szenario „Variantenstudie Anlagentechnik“ (siehe Abschnitt 1.3.1) gewählt. Andere Zuordnungen z.B. auf Simulationsmodelle mit einer detaillierten Modellierung wären ebenfalls denkbar. Man erkennt, dass die Zuordnung i.d.R. nicht über die Klassenzuordnung in IFC erfolgt, da dies nicht eindeutig wäre, sondern über die zugeordneten *Type-Objekte* und deren Attribut *PredefinedType* (Grundlagen dazu wurden in Abschnitt 2.1.5

³⁴ Bezogen auf die Gesamtmenge aller Modelica-Bibliotheks-Komponenten und IFC-Klassen handelt es sich somit um eine eindeutige Zuordnung

beschrieben). Diese Zuordnungen lassen sich der oben beschriebenen Problemklasse 2 zuzuordnen.

In Tab. 38 ist zu erkennen, dass mehreren *IFC-Type*-Gruppen dieselbe AixLib-Klasse *LosslessPipe* zugeordnet wird. Dies würde bei der Erzeugung von IFC aus Modelica (Rückweg) zu Problemen führen, da es keine *eineindeutige* Zuordnung ist. Jedoch werden die grau dargestellten Komponenten in den hier betrachteten Use-Cases Modelica-seitig ohnehin nicht berücksichtigt, da sie nicht strukturrelevant sind (Erläuterungen dazu in Abschnitt 3.5). Lässt man die grauen Zeilen außen vor, erfüllt die in Tab. 38 dargestellte Zuordnung die oben aufgestellte Anforderung nach Eineindeutigkeit.

Für Wärmemengenzähler existiert in der AixLib kein geeignetes Pendant, es wird daher ausnahmesweise Bezug genommen auf ein benutzerdefiniertes Objekt aus der sog. LibEAS (interne Bibliothek am Institut der Autorin). Wenn diese Bibliothek nicht zur Verfügung steht, muss das Alignment angepasst werden und eine alternative Bibliotheks-Komponente eingesetzt werden.

Die unterstützten Klassen in Tab. 31 ergeben sich aus der Definition des Alignments, wie in Abschnitt 3.8.4.3 beschrieben. Wird das Alignment (Tab. 38, siehe auch Abschnitt 4.3.4.1) erweitert, so werden ohne weitere Anpassung der Implementierung auch mehr Klassen unterstützt. Dies ist ein wesentlicher Vorteil der Verwendung von Wissensgraphen für den Übersetzungs-UseCase.

Tab. 38 Mapping für hydraulische Komponenten

Orange: BIM2SIM eindeutig, umgekehrt nicht, pink: SIM2BIM eindeutig, umgekehrt nicht, blau: mapping auf benutzerdefinierte Komponente

Klasse	PredefinedType	AixLib Kompoente (grau: nur relevant für SIM2BIM, falls es so modelliert wurde)	Verwendung im Bsp.
IfcPipeSegment	IfcPipeSegmentType=USERDEFINED	AixLib.Fluid.FixedResistances.LosslessPipe	Replacements
	IfcPipeSegmentType=RIGIDSEGMENT	AixLib.Fluid.FixedResistances.LosslessPipe	
IfcPipeFitting	IfcPipeFittingType=BEND	AixLib.Fluid.FixedResistances.LosslessPipe	
	IfcPipeFittingType=CONNECTOR	AixLib.Fluid.FixedResistances.LosslessPipe	
	IfcPipeFittingType=JUNCTION	AixLib.Fluid.FixedResistances.Junction	
IfcFlowInstrument ³⁵	IfcFlowInstrumentType =PRESSUREGAUGE	AixLib.Fluid.Sensors.RelativePressure	
IfcHeatExchanger	IfcHeatExchangerType =PLATE	AixLib.Fluid.HeatExchangers.ConstantEffectiveness	
	IfcHeatExchangerType =USERDEFINED	AixLib.Fluid.HeatExchangers.SensibleCooler_T AixLib.Fluid.HeatExchangers.HeaterCooler_u	Heizkreise, z.B. BKA
IfcPump	IfcPumpType=CIRCULATOR	AixLib.Fluid.Movers.FlowControlled_m_flow AixLib.Fluid.Movers.SpeedControlled_y	
IfcValve	IfcValveType =CHECK	AixLib.Fluid.FixedResistances.CheckValve	
	IfcValveType =ISOLATING	AixLib.Fluid.Actuators.Valves.TwoWayLinear	
	IfcValveType =MIXING	AixLib.Fluid.Actuators.Valves.ThreeWayLinear	
IfcTank	IfcTankType=EXPANSION	AixLib.Fluid.Sources.Boundary_pT	
	IfcTankType=STORAGE	AixLib.Fluid.Storage.Stratified	
IfcSensor	IfcSensorType=FLOWSENSOR	AixLib.Fluid.Sensors.EnthalpyFlowRate	
	IfcSensorType=HEATSENSOR	LibEAS.Sensors.WMZ	
	IfcSensorType=PRESURESENSOR	AixLib.Fluid.Sensors.Pressure	
	IfcSensorType=TEMPERATURESENSOR	AixLib.Fluid.Sensors.TemperatureTwoPort	
IfcBoiler	IfcBoilerType=WATER	AixLib.Fluid.BoilerCHP.Boiler	
IfcUnitaryEquipment	IfcUnitaryEquipmentType=SPLITSYSTEM	AixLib.Fluid.HeatPumps.HeatPump	

³⁵ *IfcFlowInstrument* dient dem Messen und Anzeigen, im Gegensatz dazu dient ein *IfcSensor* zur Übertragung der Messwerte (Aufschaltung GA).

3.8.4.4 IFC-IBPSA-Wrapper

Tab. 39 zeigt eine Mapping-Tabelle für Lüftungsanlagen. Orange hervorgehoben sind dabei Zuordnungen, die bezogen auf die AixLib, Modelica-seitig nicht eindeutig sind. Bei Lüftungsanlagen bestehen anders als bei den hydraulischen Anlagen, die im vorigen Abschnitt thematisiert wurden, keine Nebenbedingungen, die es ausschließen, dass diese Dopplungen praktisch relevant werden. Daher wird hierfür ein alternativer Lösungsansatz gewählt: Es wurde Modelica-seitig eine sog. **Wrapper-Bibliothek** angelegt, diese enthält für jede IFC-Klasse bzw. Kombination aus IFC-Klasse und predefinedType eine Komponente. Das Alignment ist damit sehr einfach auf Klassenebene zu erstellen. Die Problemklasse gemäß Abschnitt 3.8.2 wurde damit auf „0“ reduziert, der Zuordnungs-Aufwand verschiebt sich zum Ersteller der Wrapper-Bibliothek.

*Tab. 39 Mapping-Tabelle IFC Modelica für Komponenten von Lüftungsanlagen
(orange: nicht eindeutig im Rahmen der Lüftungskomponenten, blau: nicht eindeutig in Kombination mit den hydraulischen Komponenten (Tab. 38))*

IFC4.2.0.1	AixLib Kompoente
ifcFan	AixLib.Fluid.Movers.FlowControlled_m_flow
ifcAirtoAirHeatRecovery	AixLib.Fluid.HeatExchangers.ConstantEffectiveness
IfcCoil	AixLib.Fluid.HeatExchangers.ConstantEffectiveness
ifcFilter	AixLib.Fluid.FixedResistances.LosslessPipe
IfcDamper	AixLib.Fluid.FixedResistances.LosslessPipe
ifcDuctSilencer	AixLib.Fluid.FixedResistances.LosslessPipe
IfcAirTerminal	AixLib.Fluid.FixedResistances.LosslessPipe
IfcAirTerminalBox	Fluid.Actuators.Valves.TwoWayLinear

Abb. 41 zeigt einen Screenshot der dafür erstellten Wrapper- Bibliothek: Für jede relevante IFC-Type-Klasse-Kombination gemäß den Tabellen Tab. 38 und Tab. 39 ist eine Komponente enthalten. Es wären auch weitere Komponenten denkbar, die weitere Informationen gemäß der Notwendigkeit aufgrund der Problemklasse (siehe Abschnitt 3.8.2) enthalten, solche Fälle sind in den Usecases gemäß Abschnitt 1.5.1 jedoch nicht vorhanden.

The image consists of two side-by-side screenshots. On the left, a tree view of components is shown under the root 'IBPSA_Translator'. The 'IfcFlowMovingDevice' node is expanded, showing 'IfcFan' and 'IfcPump' as children. Other collapsed nodes include 'IfcDistributionFlowElement', 'IfcEnergyConversionDevice', 'IfcFlowController', 'IfcValve', 'IfcFlowTreatmentDevice', 'IfcFlowTerminal', 'IfcFlowSegment', 'IfcFlowFitting', and 'IfcDistributionControlElement'. On the right, a detailed configuration dialog for 'IfcPump' is displayed. The dialog has tabs for General, Dynamics, Initialization, Assumptions, Advanced, Add modifiers, and Attributes. The General tab shows the component name as 'ifcPump' and its path as 'IBPSA_Translator.IfcDistributionFlowElement.IfcFlowMovingDevice.IfcPump'. It includes sections for Medium (set to 'Modelica.Media.Interfaces.PartialMedium'), Nominal condition (with fields for 'm_flow_nominal' and 'dp_nominal'), and Control (with 'inputType' set to 'AixLib.Fluid.Types.InputType.Continuous'). An icon for 'IfcPump' is shown in the top right corner of the dialog.

Abb. 41 Screenshot der Wrapper-Bibliothek

Abb. 42 Screenshot der Parameter der IfcPump-Klasse in der Wrapper-Bibliothek

Die Klassendefinitionen dieser Wrapper-Komponenten sind sehr kurz - Lis. 13 zeigt die vollständige Definition der Wrapper-Komponenten für IfcPump. Die Definition enthält lediglich die Vererbungsbeziehung zu jeweils zu mappenden AixLib-Klasse gemäß den Mapping-Tabellen. Damit ist der gesamte Funktionsumfang der entsprechenden AixLib-Komponente enthalten, darunter die Icons und die Parameter (siehe Abb. 41, Abb. 42).

```
within IBPSA_Translator.IfcDistributionFlowElement.IfcFlowMovingDevice;
model IfcPump
  extends AixLib.Fluid.Movers.FlowControlled_m_flow
  annotation (Icon(coordinateSystem(preserveAspectRatio=false)), Diagram(
    coordinateSystem(preserveAspectRatio=false)));
end IfcPump;
```

Lis. 13 Klassendefinition der Wrapper-Komponente für IfcPump

3.9 TTL2MO (Konverter 5)

Für den UC1 (BIM2SIM) ist es notwendig, aus einem Modelica-Wissensgraph wieder ein natives Modelica-File zu erzeugen. Aus dem Wissensgraph *xyz_mo.ttl* mit dem AixLib und MoOnt-Vokabular muss eine native Modelica-Datei erstellt werden. Dafür wird der Konverter TTL2MO vorgesehen.

Wie in Abschnitt 2.2.3 bereits beschrieben wurde, sind MO-Dateien auch ohne grafischen Layer korrekt und für eine Simulation nutzbar. Jedoch wird für den vorliegenden Use-Case davon ausgegangen, dass kein vollautomatischer Workflow realisiert wird, sondern lediglich eine Teilautomatisierung der arbeitsreichen aber trivialen Arbeitsschritte. Im Ergebnis soll eine valide Struktur des Modelica-Modells entstehen.

Nach der Erzeugung der Modell-Struktur soll der Simulationsingenieur das Simulationsmodell weiter behandeln und Optimierungen im Anlagensystem (nicht nur an dessen Modelica-Modell) vornehmen. Daher ist es notwendig auch den Diagram-Layer zu befüllen, um eine graphische Weiterbearbeitung zu ermöglichen.

Bei Nutzung von Bibliotheks-Komponenten, welche ein Icon definiert haben, wird dieses automatisch im Diagram-Layer angezeigt. Wichtig ist jedoch die Platzierung dieser Icons, so dass es nicht zu Überlappungen der Icons und möglichst wenig Kreuzungen ihrer Verbindungen kommt. Dieselben Anforderungen bestehen bereits bei der Abbildung als Schema und wurden dafür auch gelöst (siehe Abschnitt 3.5.4). Es wird dieselbe Anordnung der Icons im Modelica-Diagram-Layer verwendet wie im Schema d.h. in der 2D-Repräsentation in der IFC-Datei. Dies gewährleistet eine gute Zuordenbarkeit zwischen Schema und Modelica-Modell und erleichtert damit die Kommunikation zwischen Planungsingenieur und Simulationsingenieur.

Verbindungen werden im Diagram-Layer nur dargestellt, wenn ihr Verlauf als Annotation im Modell enthalten ist. Es wurden der Einfachheit halber zunächst geradlinige Verbindungen zwischen den jeweiligen linken unteren Eckpunkten der Icons implementiert.

3.10 MoTTL-Transcriptor (Konverter 7)

Der MoTTL-Transcriptor dient dazu, Modelica-Files in Wissensgraphen (grüner Pfeil in Abb. 43 und Abb. 33) umzuwandeln und nutzt dazu das Vokabulars MoOnt. Wie bereits im Abschnitt "Modelica-Grundlagen" beschrieben, gibt es kein Schlüsselwort, das zwischen *Modelica-Executables* (A-Box) und *Bibliotheks-Komponenten* (T-Box), die als Komponenten für andere Modelle dienen, unterscheidet. Der Transkriptor unterscheidet daher ebenfalls nicht zwischen diesen beiden.

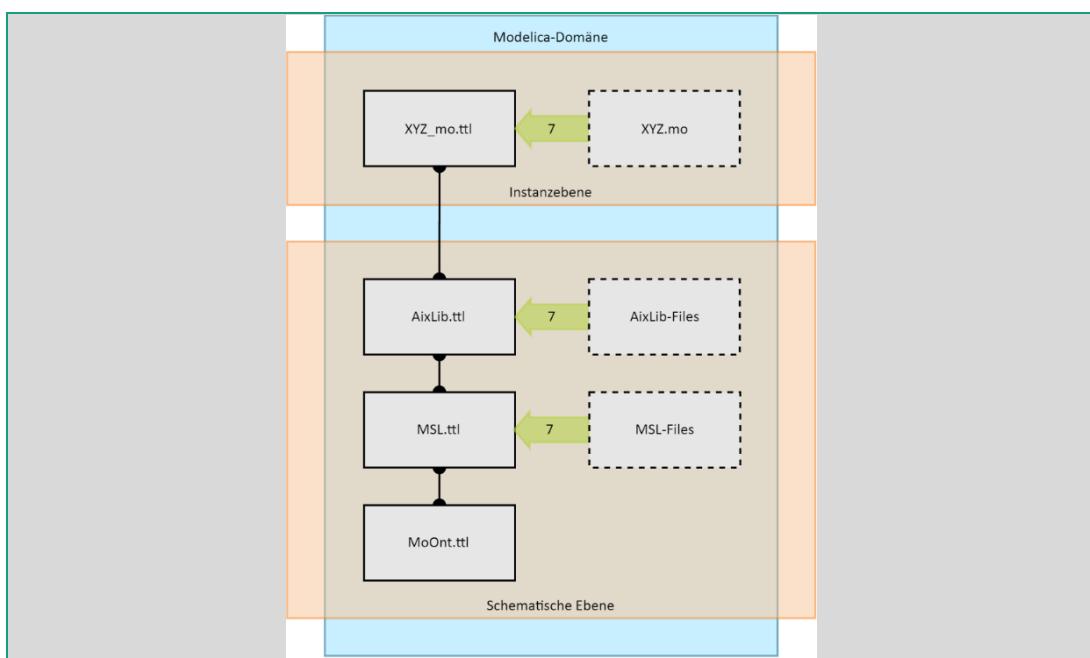


Abb. 43 Ablauf auf schematischer und Instanz-Ebene

Der MoTTL-Transcriptor überträgt nicht alle Informationen aus dem Modelica-File in den Modelica-Wissensgraph sondern lediglich jene, die gemäß den Anforderungen, welche auf S. 107 zusammengefasst wurden, nötig sind.

Die Transkription erfolgt in strikter Übereinstimmung mit der Modelica-Syntax: Klassen (*class/model*) werden als solche transkribiert (*rdfs:subClassOf*). Modelica-Komponenten werden

als Instanzen des Wissensgraph transkribiert (*a* entspricht *rdf:type*), dies ist im Lis. 14 beispielhaft dargestellt.

```
aix:Aix.Airf.AC.AirCurtainSimplified rdfs:subClassOf moont:Mmodel.
moont:MModel rdf:type owl:Class.
aix:Aix.Airf.AC.AirCurtainSimplified moont:hasPart aix:AixLib.Airflow.AirCurtain.AirCurtainSimplified.TBou.
aix:Aix.Airf.AC.AirCurtainSimplified.TBou a moont:MparameterComponent.
```

Lis. 14 Ausschnitt aus der Transkription der AixLib in einen KG (grün: statement aus MoOnt, blau: statements aus AixLib)³⁶

Am Beispiel einer Bibliotheks-Komponente (Lis. 15) und eines Modelica-Executable (Lis. 16) werden in den folgenden Listings die beiden Modellrepräsentation als Modelica-File und als Modelica-Wissensgraph gegenübergestellt. Inhaltliche Entsprechungen sind jeweils in derselben Farbe hervorgehoben. Die beiden Auszüge in Lis. 15 zeigen die Modelica-Datei des *Carnot_y*-Wärmepumpenmodells und sein Pendant im Wissensgraphen. Die *extends*- und *containedIn*-Beziehungen sind grün und orange hervorgehoben.

Natives Modelica-File:

```
within AixLib.Fluid.HeatPumps;
model Carnot_y
  "Reversible heat pump with performance curve adjusted based on Carnot efficiency"
  extends AixLib.Fluid.Chillers.BaseClasses.PartialCarnot_y(
    final COP_is_for_cooling = false);
  initial equation
    assert(COP_nominal > 1, "The nominal COP of a heat pump must be bigger than one.");
    annotation(
      defaultComponentName="heaPum",
      Documentation(info="
<p>
This is model of a heat pump whose coefficient of performance COP changes
with temperatures in the same way as the Carnot efficiency changes.
The input signal <i>y</i> is the control signal for the compressor.
</p>
...
</html>"),
      Icon(graphics={
        Line(points={{0,68},{0,90},{90,90},{90,90},{100,90}},
              color={0,0,255})));
    );
  end Carnot_y;
```

Wissensgraph:

```
mbl:AixLib.Fluid.Chillers.Carnot_y rdfs:subClassOf moont:MModel;
  moont:stringComment "Chiller with performance curve adjusted based on Carnot effi-
  ciency"^^xsd:string;
  moont:containedIn aix:AixLib.Fluid.Chillers.
aix:AixLib.Fluid.Chillers.Carnot_y moont:extends aix:AixLib.Fluid.Chillers.BaseClasses.PartialCar-
  not_y.
```

Lis. 15 Vergleich eines Ausschnitts aus dem *Carnot_y*-Wärmepumpenmodell (Bibliothekskomponente) als natives Modelica file³⁷ (oben) und als Mo-KG (unten) turtle syntax³⁸.

Die Farben dienen der Zuordnung inhaltsgleicher Abschnitte.

Man erkennt in Lis. 15, dass beispielsweise die Dokumentation des Modells, welche als Modelica *annotation* im Modell vorliegt, nicht in den Wissensgraph übertragen wurde. Ebenso wenig werden die Inhalte der *equations*-Sektion in Modelica-Files in den KG übertragen. Eine Ausnahme davon

³⁶ https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/ontologies/8_ModelicaLibraries/AixLib_1.0.0-0.8.0.ttl

³⁷ https://github.com/RWTH-EBC/AixLib/blob/development/AixLib/Fluid/HeatPumps/Carnot_y.mo

³⁸ https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/ontologies/8_ModelicaLibraries/MBL_8.0.0-0.8.0.ttl

3 Methodischer Ansatz

3.10 MoTTL-Transcriptor (Konverter 7)

3.8.4 IFC – Modelica – Alignment

bilden lediglich die *connect-equations*, wie in der Gegenüberstellung Lis. 16 erkennbar ist. Für die weitere Verwendung im UC1 sind die Informationen im Mo-KG jedoch ausreichend.

Natives Modelica-File:

```
...
AixLib.Fluid.HeatPumps.Carnot_y heaPum(
    COP_nominal=4,
    ...
    P_nominal=1000*(20/4))
...
connect (heaPum.port_b1, senT_pri_VL.port_a);
```

Wissensgraph:

```
ex:LBDCG_example.HeatPumpPlant moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.
ex:LBDCG_example.HeatPumpPlant.heaPum a moont:MComponent ;
    moont:identifier "heaPum";
    a aix:AixLib.Fluid.HeatPumps.Carnot_y.
ex:LBDCG_example.HeatPumpPlant.heaPum.COP_nominal moont:modification "4.0"^^xsd:Real;
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.COP_nominal.
...
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.P_nominal.
ex:LBDCG_example.HeatPumpPlant.heaPum.P_nominal moont:modification "5000"^^xsd:Real;
    moont:identifier "P_nominal".
...
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1.
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 a moont:MConnectorComponent.
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 moont:identifier "port_b1".
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 moont:connectedTo ex:LBDCG_example.HeatPump-
    Plant.senT_pri_VL.port_a.
```

Lis. 16 Vergleich eines Ausschnitts aus dem heatPumpPlant (Modelica Executable) als natives Modelica file³⁹ (oben) und als Mo-KG (unten) turtle syntax⁴⁰

Bei der Transkription wird jede Komponente des Modelica-Modells als eine Instanz übernommen, dies sollte bei der Vorbereitung des Modelica-Modells beachtet werden. Üblich ist beispielsweise das Ergänzen von Regelungs- und Auswertungskomponenten (meist aus den Bibliotheken *Modelica.Blocks*, *AixLib.Controls*, *Buildings.Controls*) auf oberster Modellebene. Dies entspricht jedoch nicht der Realisierung in der Praxis, wo dementsprechende Komponenten als Software in einem Hardware-Controller realisiert werden. Soll der entstehende Mo-KG für eine Massenermittlung (wie in (Eckstädt, Menzel, u. a. 2023) beschrieben) genutzt werden, so ist das Modell geeignet aufzubereiten. Abb. 44 zeigt zwei Varianten desselben Modells. In einem für die Massenermittlung geeigneten Modell entspricht jede Modelica-Connection einem realen Bauteil. Im gezeigten Beispiel entsprechen hellblaue Verbindungen Rohren bzw. Kanälen und dunkelblaue bzw. gelbe Verbindungen werden in Realität durch Signalkabel realisiert.

Für den UC2 SIM2BIM bestehen hingegen keine speziellen Anforderungen an das Modell. Regelung- und Auswertungskomponenten sind nicht Bestandteil des Alignment (siehe Tab. 38). Sie werden daher nicht nach IFC-übersetzt – ebensowenig wie die zugehörigen Verbindungen.

³⁹ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2GA/C_HeatPumpPlant/LBDCG_example/HeatPumpPlant.mo

⁴⁰ https://github.com/ElisEck/MO-x-IFC/blob/public-main/src/test/java/output/ex_20221215_1154_fullclean.ttl

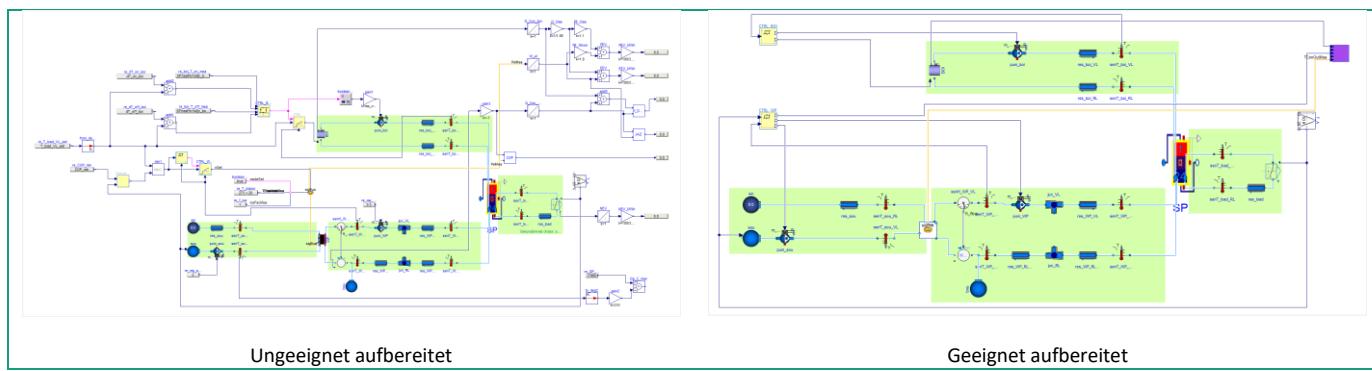


Abb. 44 Gegenüberstellung eines für die Massenermittlung ungeeignet und geeignet aufbereiteten Modelica-Modells

3.11 ProTran Procedural Translator (Konverter 9)

Die Vorteile des Semantic Reasoning wurden auf Seite 114 ausführlich beschrieben, in der Literatur (Kralin 2019; Feeney 2019) werden jedoch auch Nachteile des Reasoning beschrieben:

- Laufzeitprobleme
- Hohe Anforderungen an Arbeitsspeicher
- Komplizierte und fehleranfällige Formulierung der Alignments (First Order Logic Wissen erforderlich) und logische Zusammenhänge mittels OWL ist schwierig
- Technische Beschränkung in Hinblick von *DataTypeProperties*

Wird der Teilschritt „reasoning“ durch gezielte SPARQL queries ersetzt, entfallen diese Nachteile, jedoch wird die Implementierung dann richtungsabhängig und wird daher als „Procedural Translation“ bezeichnet. Abb. 45 stellt Semantic Translation und Procedural Translation als zwei Implementierungsvarianten für die Translation-Komponenten gegenüber.

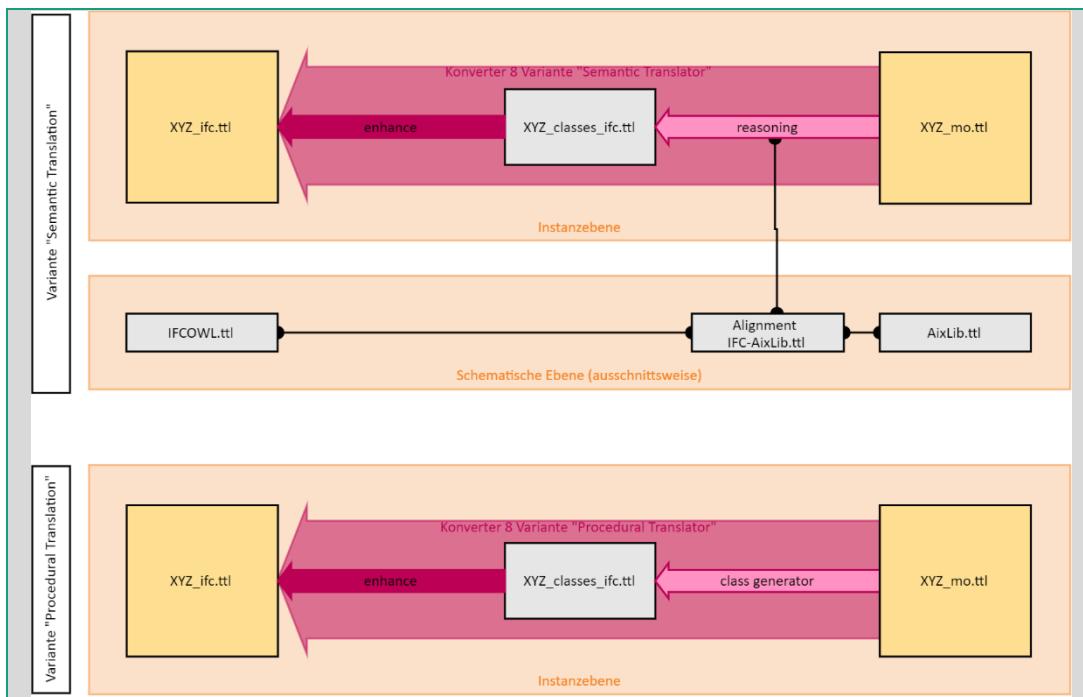


Abb. 45 Gegenüberstellung „Semantic Translation“ zu „Procedural Translation“ für UC4 SIM2BIM BRICK

Die Informationen bzgl. der Zuordnung von Entitäten in beiden Domänen sind nach wie vor notwendig. Bei der Implementierung als *Procedural Translation* werden diese Mapping-Tabellen

jedoch nicht als Alignment formuliert, sondern werden im Quelltext für die Erzeugung der SPARQL-queries hinterlegt (siehe Lis. 30 in Abschnitt 4.3.7).

Es werden bei dieser Transkription (unabhängig davon, ob sie als Semantic oder Procedural Translation implementiert würde) jedoch keine Tripel für eine geometrische Repräsentation erzeugt, da dies den IFC-KG unnötig vergrößern würde. Die Basisinformationen (*IfcDistributionPort* und *IfcRelConnectsPorts*) für eine schematische 2D-geometrische Repräsentation sind jedoch im erzeugten IFC-KG enthalten.

3.12 Voluminizer (Konverter 10)

Wie in Abschnitt 1.1.2 bereits erläutert wurde, hat das Funktionsmodell keine Geometrieinformationen. Das Simulationsmodell (als Spezialfall des Funktionsmodells) enthält dementsprechend auch keine Geometrieinformationen über die Platzierung der Symbole in der schematischen Darstellung hinaus. Für das Handling in IFC-Viewern und für die Weiterverarbeitung von IFC-Modellen mit Autorentools ist eine 3D-Geometrie i.d.R. notwendig, da diese oftmals keine zweidimensionalen Repräsentationen unterstützen (siehe Tab. 6). Die durch die Translation erzeugte zweidimensionale Repräsentation in der IFC-STEP-Datei soll mit dem Postprocessing-Konverter *Voluminizer* daher um eine dreidimensionale Geometrie ergänzt werden. Es werden dafür stark vereinfachte Platzhalter-Geometrien vorgesehen, die Platzierung erfolgt an der Stelle des Icons – für alle übersetzten Entitäten in einer Ebene.

Um eine gute Weiterverarbeitbarkeit zu gewährleisten, werden die Platzhalter so skaliert, dass sie in einer dreidimensionalen Ansicht gut auswählbar sind. Die Klasse-Typ-Zuordnung wird mit Form und Farbe kodiert.

4 Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web

In diesem Kapitel werden zunächst die Anforderungen an die Implementierung konkretisiert. Die Implementierung erfolgt mit open-source-Tools als sog. „MO-x-IFC-Toolset“. Dieses besteht aus verschiedenen Software-Komponenten sog. „MOXIK“ (MO-x-IFC-Konverter), welche sich in die drei bereits in Kapitel 3 eingeführten Gruppen Pre- und Postprocessing, Transkription sowie Translation einteilen lassen.

Zunächst werden die Anforderungen an die zu verarbeitenden IFC-Dateien beschrieben. Ausführlich werden anschließend Wege aufgezeigt, diese herzustellen, falls sie nicht erfüllt sind. Dazu wurden eine Reihe von Skripten entwickelt.

Die notwendigen acht MOXIK zur Realisierung beider UseCases werden in den Abschnitten 4.3.2 bis 4.3.9 im Einzelnen beschrieben. Für die Translationskomponenten werden zwei verschiedene Implementierungen vorgesehen (Semantic und Procedural).

4.1 Anforderungen an die Implementierung

Auch die Implementierung des Toolsets soll mit Open-Source-Komponenten erfolgen. Dies gewährleistet die wissenschaftlich notwendige Nachvollziehbarkeit und soll auch Vertrauen bei potentiellen Nutzern schaffen. Das Toolset soll bewusst *keine* Black-Box darstellen, der blind vertraut werden muss.

Die Verwendung von Open Source Komponenten ermöglicht ein hohes Qualitätsniveau der Implementierung. Weiterhin senkt es finanzielle Hürden bei potentiellen Nutzern, da hier kein Investment getätigt werden muss. Die Entwicklung aller Komponenten erfolgte, wenn nicht anders angegeben, mit *Python 3.8* unter Windows.

Die eindeutigen Identifikatoren in den verschiedenen Sprachen (*GlobalID* in IFC, *identifier* in Modelica), sollen – einmal vergeben – durch alle Schritte des Workflows erhalten werden.

4.2 Übersicht über die Konverter des MO-x-IFC-Toolsets

Tab. 40 gibt einen Überblick über die implementierten Bestandteile des MO-x-IFC-Toolsets, die sog. Konverter (MOXIK). Diese werden, wie in Abschnitt 3.1 eingeführt, in 3 Gruppen eingeteilt und in den folgenden Abschnitten gemäß Tab. 40 beschrieben.

Tab. 40 Übersicht MO-x-IFC-Konverter (MOXIK)

Nr. in Spalte 21 bezieht sich auf die Pfeile in der Workflow-Übersicht in Abb. 33 auf Seite 95

	Nr	Kürzel	Kurzbezeichnung	Kap 3	Kap 4	Pfad zum Skript bzw. zur Funktion Basispfad https://github.com/ElisEck/MO-x-IFC/tree/main/ (wenn nicht anders angegeben)	Kap 5
Pre- und Postprocessing	1	-	Anforderungen an IFC erfüllen	3.5.1			5.1.3
			IFC2x3-IFC4		4.3.1.1	IFC.Change.replace_ifcRelConnectsPortToElement_with_ifcRelNests*	
			Klassen und Typen		4.3.1.2	/_Projekte/EAS_KLT/08_ChangeClassesAndTypes.py*	
			Verbindungen		4.3.1.3	/_Projekte/EAS_KLT/10_AddEdgesFromManualAssignmentList.py* Wesentliche Funktionen <ul style="list-style-type: none">• IFC.IfcGraph.IfcGraph.figure_node_versions_2• IFC.IfcGraph.IfcGraph.figure_leafs_names_So• IFC.IfcGraph.IfcGraph.export_to_gephi_with_guid_labels_with_positions• IFC.IfcGraph.IfcGraph.add_edges_from_manual_assignment_list_Am2_4	
			Port-Namen		4.3.1.4	/_Projekte/EAS_KLT/26_givePortNames.py*	
			Containment		4.3.1.5	/_Projekte/EAS_KLT/13_fixContainment.py*	
			System		4.3.1.6	-	
			Benamung		4.3.1.7	/_Projekte/EAS_KLT/07_GiveDummyNames.py*	
			Verkleinerung		4.3.1.8	/02_Schematizer/02c_RemoveAllGeometry.py	
			Schematizer	Schematisierer	3.5	4.3.2	
Transkription	2	Schematizer	Vereinfachung Modell	3.5.3			5.1.3.2
			Platzierung 2D	3.5.4		/02_Schematizer/02a_Schematizer.py	
			Abbildung in IFC			/02_Schematizer/02b_Add2DGeometry.py	
			Erzeugung von 3D-Platzhaltergeometrien	3.12	4.3.3	/10_Voluminizer/10_Voluminizer.py	5.2.3.2
	3	IFC2RDF	Überführung IFC in KG	-	4.3.3	/03_IFC2RDF/IFCtoRDF-0.4-shaded.jar	
	9	RDF2IFC	Überführung von KG in IFC-STEP	-	4.3.8	https://github.com/ElisEck/RDF2IFC	
	5	TTL2MO	MoTTL-Descriptor	3.9	4.3.5	/05_TTL2MO/05_TTL2MO.py	
Translate	7	MoTTL	MoTTL-Transcriptor	3.10	4.3.8	/MoTTL-Transcriptor	
	4	SemTran	Semantic Translator	3.7			
			Alignment	3.8.4	4.3.4.1	/TBox/alignments/AlMAix_0.5.0.ttl	
			Reasoning	3.8.2	4.3.4.2	/04_SemTran/04_SemanticTranslator.py	
			Enhance_mo		4.3.4.3		
	7	ProTran	Procedural Translator	3.11	4.3.7	/08_ProTran/08_ProceduralTranslator.py	5.2.3.2

* Basispfad: <https://gitlab.cc-asf.fraunhofer.de/iis-eas-acs/ifcscripting/>

Die MOXIK stehen i.d.R. als Python-Skripte oder Funktionen zur Verfügung. Es wurde keine Benutzeroberfläche dafür vorgesehen, stattdessen wurde in eine möglichst gute Dokumentation des Codes investiert.

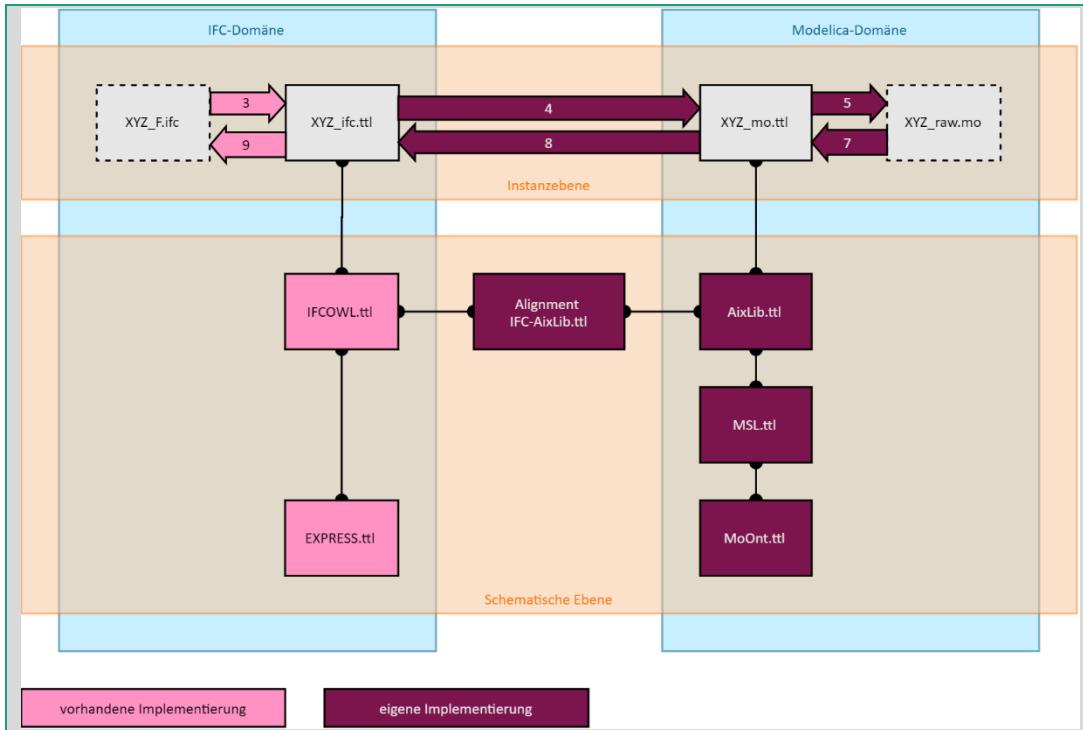


Abb. 46 Komponenten und Wissensgraphen – vorhandene und neue Implementierung

4.3 Details zu den einzelnen Konvertern MOXIK

4.3.1 IFC-seitige Voraussetzungen herstellen (Schritt 1)

Es wurde eine Reihe von Skripten entwickelt, um die in Abschnitt 3.5.1 erläuterten Voraussetzungen ggf. herzustellen:

1. IFC4
2. Korrekte Klassenzuordnung
3. Korrekte Typisierung
4. Verbindungen mit Hilfe von IfcDistributionPorts (geschlossene Anlagengraphen)
5. Portbezeichnungen
6. Zuordnung zu Geschosse
7. Zuordnung zu Systemen
8. Kurze eindeutige menschenlesbare Namen der IfcProducts
9. Kleine Dateigröße

Die Punkte 1 bis 5 sind obligatorisch, 6 bis 9 hingegen optional. Letztere erleichtern die Arbeit mit den Dateien z.B. zur Schaffung der Voraussetzung gemäß Punkt 1 bis 5.

4.3.1.1 Umwandlung IFC2x3 in IFC4

Obwohl es bereits seit Jahren den aktuelleren Standard IFC4 gibt, findet das IFC2x3-Datenschema praktisch nach wie vor eine breite Anwendung. Die Umwandlung des Datenschemas von IFC2x3 zu IFC4 ist durch die Manipulation der Angabe im Dateiheader jedoch einfach möglich, da IFC4 im Wesentlichen eine Erweiterung des IFC2x3-Schemas darstellt und das Dateiformat IFC-STEP unverändert geblieben ist. Lis. 17 zeigt den entsprechenden Ausschnitt aus der IFC-STEP-Datei.

```

ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]', '2;1'));
FILE_NAME('','2023-06-21T16:47:02',(),(), 'IfcOpenShell 0.7.0', 'IfcOpenShell
0.7.0','');
FILE_SCHEMA(( 'IFC4' ));
ENDSEC;
DATA;

```

Lis. 17 Beispiel Header-Abschnitt einer IFC-STEP-Datei

Die einzige für den vorliegenden Anwendungsfall relevante Änderung im IFC4-Datenschema besteht in der Zuordnung von *IfcDistributionPorts* zu *IfcProducts*: Hier muss die Relation *IfcRelConnectsPortToElement* durch *IfcRelNests*-Relationen ersetzt werden.

4.3.1.2 Korrektur der Klassen und Typen

Viele Ifc-Modelle enthalten *IfcBuildingElementProxys*. Diese Entität dient per Definition der Abbildung von Objekten, für welche keine geeignete Klasse im Datenschema vorhanden ist. Bei Abbildung von Anlagenstrukturen mit dem Datenschema IFC2x3, in dem viele der notwendigen Klassen noch nicht definiert waren, ist es daher wahrscheinlich, derartig klassifizierte Objekte vorzufinden. Anhand ihrer Attribute (z.B. *PredefinedType* und *Name*), *Properties* und Zuordnungen ist es i.d.R. möglich die korrekte IFC4-Klasse zu ermitteln.

Selbiges gilt für zu allgemein definierte Entitäten: In IFC2x3 existiert z.B. lediglich die Entität *IfcFlowFitting*, diese sollte durch die in IFC4-vorhandenen konkreteren Klassen wie z.B. *IfcPipeFitting* ersetzt werden.

Weiterhin können anhand der Eigenschaften der Objekte i.d.R. die passenden Type-Objekte erzeugt bzw. zugeordnet werden. Es wurde eine Funktion entwickelt, mit welcher die aktuelle Klassen- und Type-Zuordnung der Entitäten, sowie deren Eigenschaften als Excel-Tabelle exportiert werden können. Dies kann überarbeitet und anschließend wieder importiert werden.

4.3.1.3 Nachpflege fehlender Verbindungen

Korrekte Verbindungen zwischen den *IfcProducts* einer TGA-Anlage sind die am schwierigsten herzustellende Voraussetzung, wenn die Verbindungen nicht nativ vom Autorentool korrekt exportiert wurden. Eine Analyse von ca. 64 der Autorin zugänglichen IFC-Modellen von Anlagentechnik zeigte jedoch, dass *IfcDistributionPorts* in ca. 90% dieser Modelle enthalten sind. Dabei stammen die untersuchten Modelle aus unterschiedlichen Autorentools und sind teilweise auch schon sehr alt (2016) – es ist also nicht notwendig dafür ein modernes Autorentool zu benutzen.

Die Existenz von *IfcDistributionPorts* und ihrer Verbindungen ist jedoch nicht gleichbedeutend mit flächendeckenden korrekten Verbindungen im Modell. Um den vorhandenen Graph zu analysieren wurde ein Plot-Skript entwickelt. Abb. 53 zeigt ein Beispiel dafür.

Sollen basierend darauf Verbindungen entfernt und /oder ergänzt werden, steht dafür ebenfalls ein Skript zur Verfügung, welches auch ggf. fehlende *IfcDistributionPorts* an den *IfcProducts* ergänzt. Für die Nachpflege sind kurze menschenlesbare Bezeichnungen der zu verbindenden *IfcProducts* hilfreich. Diese können mit dem in Abschnitt 4.3.1.7 beschrieben Skript erzeugt werden.

4.3.1.4 Vergabe von Port-Namen

IfcDistributionPorts müssen eine eindeutige und Modelica-kompatible Bezeichnung haben: Bei Elementen mit zwei hydraulischen Anschlüssen, müssen diese *port_a* und *port_b* heißen, andernfalls sollen die Ports durchnummieriert sein: *port_1*, *port_2* und *port_3*. Wie in Abschnitt 3.5.4.3 bereits diskutiert wurde, ist es hierfür nicht notwendig eine Strömungsrichtung zu

berücksichtigen, daher werden die Bezeichnungen in der (zufälligen) Reihenfolge der Auslesung aus der IFC-STEP-Datei vergeben.

4.3.1.5 Korrektur Containment

Hilfreich für die menschliche Bearbeitung des Systems ist eine korrekte Zuordnung der IfcProducts zur kleinstmöglichen topologischen Gebäudestruktureinheit (siehe Abschnitt 2.1.9.2). Da Räume in vielen Anlagenmodellen nicht vorhanden sind, wird dies i.d.R. das Geschoss sein. Eine Nachpflege ist basierend auf der Platzierung der *IfcProduct* i.d.R. problemlos möglich. Es wurde dafür ein Skript entwickelt. Alternativ ist dies z.B. mit dem IFC-Viewer simpleBIM möglich.

4.3.1.6 Zuordnung zu Systemen

Für die Visualisierung in IFC-Viewern, aber auch für die Verkleinerung von IFC-Modellen, ist eine korrekte Systemzuordnung der IfcProducts hilfreich. Dafür ist es zunächst notwendig die Systemgrenzen zu definieren. Denkbar wäre hier zunächst hydraulische Trennungen an Wärmeübertragern als Systemgrenze zu nutzen. Dies würde in den meisten Gebäuden lediglich zwei oder drei Systeme (Heizmedium, Kühlmedium, Rückkühlmedium) führen und die Zuordnung brächte nicht viel Mehrwert. Daher wird stattdessen jeder Heiz- bzw. Kühlkreis als eigenes System definiert. Es ist auf eine einheitliche Schnittstelle zu achten, geeignet sind z.B. die Absperrorgane an den Verteilerabgängen.

Diese Zuordnung wurde interaktiv mit dem IFC-Viewer simpleBIM vorgenommen, alternativ wäre die Implementierung eines Skripts, z.B. basierend auf dem Netzwerkgraph, möglich.

4.3.1.7 Vergeben von Namen

Menschenlesbare Namen sind insb. für die Nacharbeit fehlerhafter Verbindungen hilfreich. Es wurde ein Skript implementiert, welches Namen basierend auf Klassen- und Etagenzuordnung vergibt. Zusätzlich wird eine dreistellige laufende Nummer als Suffix vergeben, um die Zuordnung eindeutig zu gestalten. Namen werden dabei nur für *IfcProduct* vergeben, welche bisher keinen Namen aufweisen. (In Abschnitt 2.1.4.1 wurde gezeigt, dass der Name ein optionales Attribut ist.)

4.3.1.8 Verkleinerung der IFC Modelle

Für das Handling der IFC-Dateien ist es notwendig die Dateien möglichst klein zu halten, andernfalls wird die Bearbeitung langsam oder ganz unmöglich. Größenbegrenzungen verschiedener Tools zeigt Tab. 58. Bei der Überführung der Modelle in einen KG werden die Dateien deutlich (ca. Faktor 10) größer. Dies wurde im Abschnitt 3.10 am Beispiel von Modelica-Dateien gezeigt, gilt jedoch in gleichem Maß auch für IFC-Dateien. Dementsprechend ist es umso wichtiger die zugrunde liegenden IFC-Dateien klein zu halten.

CAD-Autorentools erzeugen nicht unbedingt effiziente IFC-Repräsentationen. Durch alternative Serialisierungen können die Dateien ohne Informationsverlust wesentlich verkleinert werden. Dafür sind insb. Ressourcenentitäten interessant (jene STEP-Instanzen, die nicht von *IfcRoot* erben). Ein wesentlicher Hebel dafür ist die Rundung der Koordinaten auf mm, da feinere Auflösungen im Bausektor ohnehin nicht zur Anwendung kommen, von den Autorensoftwaren jedoch zum Teil ausgegeben werden. Basierend darauf können Duplikate in den *IfcCartesianPoint* erkannt und reduziert werden. Tab. 48 auf S. 151 (Schritte 1 und 2) zeigt, dass allein dadurch eine Verkleinerung auf ca. ein Drittel der ursprünglichen Größe der IFC-STEP-Datei möglich ist.

Auch nichtgeometrische IFC-Entitäten können inhaltliche Duplikate sein. Eine Bereinigung ist z.B. mit dem Solibri Optimizer (siehe Tab. 54) möglich. *IfcRoot*-Entitäten sind davon nicht betroffen, auch dabei werden ausschließlich Ressourcenentitäten verändert.

Sind die derart optimierten IFC-Dateien immer noch zu groß, kann ein systemweiser Export erfolgen. Dieser ist gegenüber dem oft praktizierten etagenweisen Export zu bevorzugen, da für

die in dieser Arbeit thematisierten UC möglich geschlossene Anlagengraphen (siehe obige Voraussetzung 4) in der IFC-Datei von erheblicher Bedeutung sind. Der systemweise Export bietet sich insb. bei komplexen Modellen wie dem in Abb. 47 gezeigten Beispiel an. Alle weiteren Schritte erfolgen dann ebenfalls systemweise, was auch die Größe des entstehenden Modelica-Modells reduziert. Die Modelica-seitigen Begrenzungen wurden in Abschnitt 3.5.2 beschrieben.

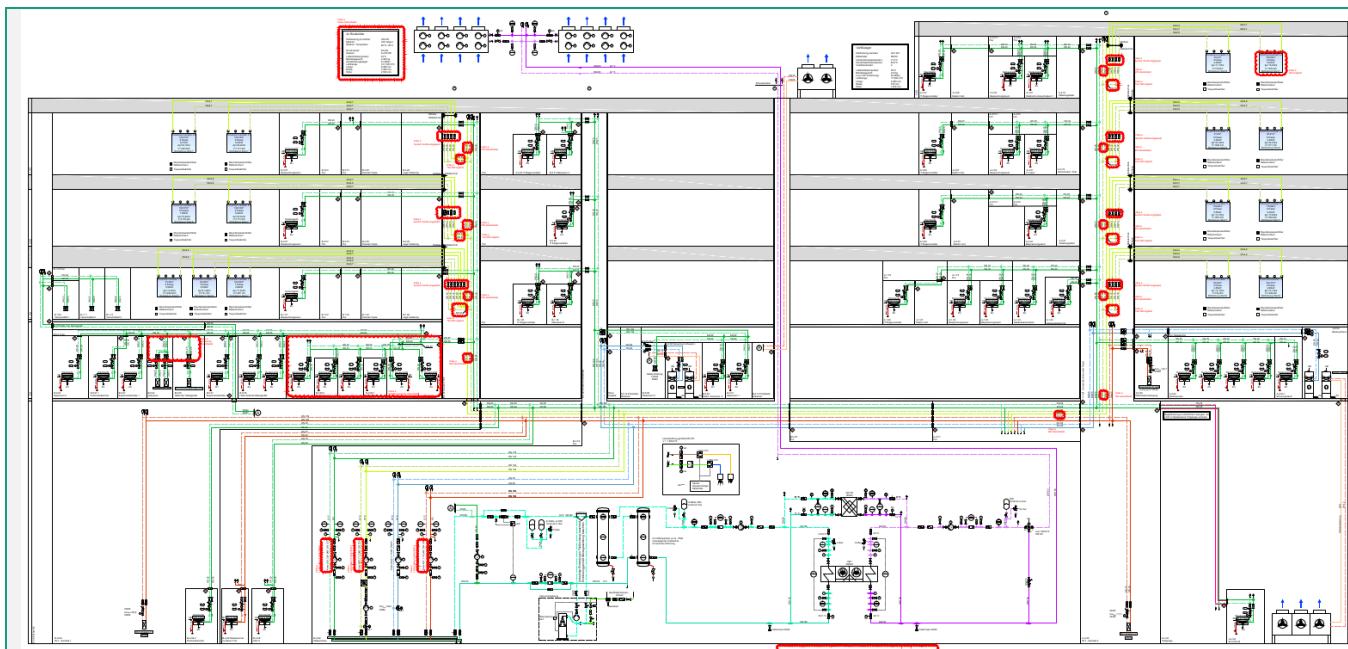


Abb. 47 Kälteschema Neubau EAS: Lesbarkeit erst ab Blatthöhe 840 (A0-Querformat und in Überbreite)

4.3.1.9 Hilfsfunktionen

Im Zuge der Modellaufbereitung wurden zahlreiche Hilfsfunktionen implementiert, insbesondere für die Handhabung der Anlagengraphen in der IFC-Datei. Diese sind im Modul *IfcNetwork* enthalten und für die Nachnutzung dokumentiert. Weiterhin sind dort Funktionen zum Entfernen geometrischer Repräsentationen (2D, 3D) implementiert, da diese für den Übersetzungs-UseCase nicht notwendig sind und sich durch ihre Entfernung die IFC-STEP-Dateien erheblich verkleinern lassen.

4.3.2 Schematizer - Abbildung der Vereinfachung in IFC (Konverter 2)

Allgemeines

Die ggf. verkleinerten und ansonsten alle Anforderungen gemäß der Aufzählung auf S. 127 erfüllenden IFC-Dateien werden anschließend mit dem Schematizer weiterverarbeitet. Dessen grundsätzliche Funktionsweise wurde in Abschnitt 3.5 erläutert, in diesem Abschnitt wird die Abbildung des Schemas im IFC-Format beschrieben.

Der Schematizer wurde als Python-Skript implementiert und steht unter folgendem Pfad zur Verfügung:

Replacements im IFC

Es ist möglich die vom Ersetzungsalgorithmus (Abschnitt 3.5.2) erzeugten *replacements* standardkonform im IFC-Format abzubilden. Dafür wurde Klasse *IfcPipeSegment* und eine Typisierung als *IfcPipeSegmentType USERDEFINED* gewählt.

Für jedes Replacement wird eine *IfcGroup* (siehe Abschnitt 2.1.9.3) angelegt, der alle ersetzen Elemente zugeordnet werden, somit kann visualisiert und geprüft werden, was ersetzt wurde. Tab. 6 zeigt welche IFC-Viewer das *IfcGroup*-Feature unterstützen. Abb. 57 zeigt ein Beispiel für die Visualisierung eines Replacements.

Für die Handhabung in IFC-Tools ist es i.d.R. notwendig, dass die Objekte eine Geometrie haben, um diese in einer 3D-Ansicht auswählen zu können. Daher wurde jedem Replacement die Geometrie eines der ersetzen Elemente zugewiesen.

Zuweisung einer 3D-Geometrie zu Replacements

Die übliche Darstellung von Schemata ist jedoch zweidimensional wie beispielhaft in Abb. 52 gezeigt. In Abschnitt 2.1.9.3 wurde erläutert, dass IFC die Speicherung verschiedener Repräsentationen ein und desselben Objekts in verschiedenen Darstellungskontexten vorsieht. Diese Funktion wird hier genutzt und ein zweidimensionaler Darstellungskontext zusätzlich in der Datei hinterlegt. Dieser wird im Folgenden als IFC-Schalschema bezeichnet.

Zuweisung einer 2D-Geometrie

Dabei werden für jedes Objekt ein Symbol (derzeit realisiert: Kreis), sowie für jede Verbindung *IfcRelConnectsPorts* eine gerade Linie dargestellt.

Die zweidimensionale Darstellung muss für eine gute Handhabbarkeit überlappungsfrei und in ähnlichem Maßstab wie die 3-dimensionale Darstellung im IFC-File enthalten sein. Es wird daher zunächst ein Layout mit dem kraftbasierten Alorithmus *neato* (siehe Tab. 34) erzeugt. Das erzeugte Layout wird anschließend noch verschoben und skaliert, um es sinnvoll zur 3D-Repräsentation der Objekte auszurichten. Tab. 41 fasst dafür die Randbedingungen zusammen.

Tab. 41 Skalierung und Platzierung der verschiedenen Layouts

Bezeichnung Layout	Platzierung	Skalierung
Original vom Layoutingalgorithmus	Am Koordinatenursprung	Von (-1 -1) bis (1 1)
Skalierung und Platzierung in IFC	In der Grundrissebene, so dass er in der „Draufsicht“ liegt, Überlappungsfrei zum Modell	Ähnliche Abmessung wie Modell (Bounding Box aller Elemente)
Skalierung und Platzierung in Modelica	Linke untere Ecke entspricht Koordinatenursprung (0 0)	Verfügbares Canvas entspricht Auflösung aktueller Bildschirme → ca. 1000x2000

Für eine einfachere Handhabbarkeit im TTL2MO-Konverter werden an jedem *IfcElement* die Zentrumskoordinaten aus der 2D-Repräsentation zusätzlich als *IfcProperty* ergänzt, denn diese sind für die Platzierung im Simulationsmodell ebenfalls relevant.

Finalisierung des Funktionsmodells

Es steht eine Funktion zur Verfügung, um ersetzte *IfcProducts* aus der Datei zu entfernen, sollte die für die Reduzierung der Dateigröße nötig sein. Unabhängig davon, kann auch nur die geometrische Repräsentation der *IfcProducts* entfernt werden.

4.3.3 IFC2OWL Überführung IFC-STEP in Wissensgraph (Konverter 3)

Bei der Überführung einer IFC-STEP-Datei in ein IFC-OWL-File ist eine deutliche Vergrößerung der Dateien zu erwarten, da das TTL-Format ganz allgemein spezifiziert und nicht für die Darstellung von Produktdaten (wie das STEP-Format⁴¹) optimiert ist. Beispielhaft erkennt man dies im Vergleich von Lis. 18 und Lis. 19.

Um innerhalb der Grenzen der Handhabbarkeit zu bleiben (siehe Tab. 58), werden vor der Umwandlung auch noch die verbliebenen 3D-Geometrien entfernt. Auch die 2D-Geometrien sind unnötig, beanspruchen jedoch sehr wenig Speicher und werden daher erhalten.

```
#364=IFCPUMP('2lqrrghQz070000001nNz',#162,'FLOWMOVINGDE-
VICE(1)',$,',',#362,#363,$,$);
#1354=IFCPUMPTYPE('OnyWrvjrf3FR$BZmA4v$qt',#162,'65 mm Durchm. Inline-Einfach-
blockpumpe',$$,($#562),$$,$,$,.CIRCULATOR.);
```

Lis. 18 Ausschnitt aus Beispiel-IFC B25.ifc (STEP-Syntax)

⁴¹ Das STEP Format wiederum ist auch nicht optimiert für die Darstellung von 3D Geometrie und daher im Vergleich zu z.B. Revit-Dateien „umständlicher“.

4 Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web

4.3 Details zu den einzelnen Konvertern MOXIK

4.3.4 SemTran Semantic Translator (Konverter 4)

```
inst:IfcPump_364 rdf:type ifc:IfcPump .
inst:IfcPump_364 ifc:globalId_IfcRoot inst:IfcGloballyUniqueId_1930 ;
  ifc:ownerHistory_IfcRoot inst:IfcOwnerHistory_162 ;
  ifc:name_IfcRoot inst:IfcLabel_1931 ;
  ifc:objectType_IfcObject inst:IfcIdentifier_1429 ;
  ifc:objectPlacement_IfcProduct inst:IfcLocalPlacement_362 ;
  ifc:representation_IfcProduct inst:IfcProductDefinitionShape_44 .
inst:IfcGloballyUniqueId_1930
  rdf:type ifc:IfcGloballyUniqueId ;
  express:hasString "2lqrrghQz0700000001nNz" .
inst:IfcLabel_1931 rdf:type ifc:IfcLabel ;
  express:hasString "FLOWMOVINGDEVICE(1)" .
inst:IfcRelDefinesByType_1410
  ifc:globalId_IfcRoot inst:IfcGloballyUniqueId_2982 ;
  ifc:ownerHistory_IfcRoot inst:IfcOwnerHistory_46 ;
  ifc:relatedObjects_IfcRelDefinesByType inst:IfcPump_364 ;
  ifc:relatingType_IfcRelDefinesByType inst:IfcPumpType_1354 .
inst:IfcPumpType_1354
  rdf:type ifc:IfcPumpType .
inst:IfcPumpType_1354
  ifc:globalId_IfcRoot inst:IfcGloballyUniqueId_2924 ;
  ifc:ownerHistory_IfcRoot inst:IfcOwnerHistory_162 ;
  ifc:name_IfcRoot inst:IfcIdentifier_2410 ;
  ifc:hasPropertySets_IfcTypeObject inst:IfcPropertySet_562 ;
  ifc:predefinedType_IfcPumpType ifc:CIRCULATOR .
inst:IfcGloballyUniqueId_2924
  rdf:type ifc:IfcGloballyUniqueId ;
  express:hasString "0nyWrvjrf3FR$BZmA4v$qt" .
```

Lis. 19 Ausschnitt aus Beispiel IFC als KG B25.ttl (Turtle-Syntax), dieser umfasst einen Teil der Informationen in Lis. 18

Für die Überführung von IFC-STEP-Dateien in IFC-OWL-Files wird der Konverter *IFCtoRDF* von (siehe Tab. 54) verwendet. Er steht als Kommandozeilenanwendung zur direkten Nutzung, sowie als „Maven dependency“ zur Verwendung in eigenen Java-Anwendungen zur Verfügung.

Anschließend erfolgt die Umwandlung mittels des in Lis. 20 dargestellten Aufrufs. Als Eingabeparameter sind der Pfad zur IFC-STEP-Datei sowie der Pfad zum zu erzeugenden KG anzugeben. Dieser KG wird als ttl-File serialisiert.

```
Java -jar IFCtoRDF-0.4-shaded.jar xyz_F.ifc xyz_ifc.ttl
```

Lis. 20 Aufruf der IFCtoRDF Kommandozeilenanwendung

Das Skript für die Entfernung der 3D-Geometrie und die *IfctoRDF*-Anwendung werden im MO-x-IFC-Toolset zusammen als *IFC2OWL*-Konverter bezeichnet.

4.3.4 SemTran Semantic Translator (Konverter 4)

4.3.4.1 Alignment

Es wurde zunächst das Alignment zwischen IFC und AixLib gemäß Tab. 38 als Wissensgraph erstellt. Dieses enthält hauptsächlich *PropertyChains* und Klassen-Äquivalenzbeziehungen *owl:equivalentClass*. Insgesamt wurden 177 Axiome vorgegeben, es wurde der NameSpace <https://w3id.org/aimaix/> mit dem Präfix *aimaix:* verwendet. Die Erstellung erfolgte manuell im Protege-Editor. Lis. 21 gibt beispielhaft die Triples für die Zuordnung von Umwälzpumpen in IFC und AixLib im Alignment wieder.

```

@prefix aimaix: <https://w3id.org/aimaix/> .
@prefix aix: <https://w3id.org/aix/> .

aimaix:P_C rdf:type owl:Class ;
    owl:equivalentClass aix:AixLib.Fluid.Movers.FlowControlled_m_flow ,
        [ rdf:type owl:Restriction ;
            owl:onProperty ifc:isTypedBy_IfcObject ;
            owl:someValuesFrom aimaix:IfcRelDefinesByType_P_C
        ] .
aimaix:IfcRelDefinesByType_P_C rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
        owl:onProperty ifc:relatingType_IfcRelDe_finesByType ;
        owl:someValuesFrom aimaix:IfcPumpType_CIRCULATOR
    ] .

aimaix:IfcPumpType_CIRCULATOR rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
        owl:onProperty ifc:predefinedType_IfcPumpType ;
        owl:hasValue ifc:CIRCULATOR
    ] .

```

Lis. 21 Ausschnitt aus Alignment zur Zuordnung von Umwälzpumpen in IFC und AixLib (ttl-Syntax)⁴²

Es wurden drei neue Klassen im Namespace des Alignments *aimaix* definiert, um die Lesbarkeit und Nachvollziehbarkeit zu erhöhen. Diese werden in Tab. 42 erläutert. Lis. 22 zeigt beispielhaft die passenden Triple zum Alignment in Lis. 21 aus dem IFC-KG des Beispiels in Abschnitt 5.1.

```

@prefix inst: <http://linkedbuildingdata.net/ifc/resources20230725_090232/> .
inst:IfcPumpType_1354
    ifc:predefinedType_IfcPumpType ifc:CIRCULATOR .
inst:IfcPumpType_1354
    rdf:type ifc:IfcPumpType .
inst:IfcRelDefinesByType_1410
    ifc:relatedObjects IfcRelDefinesByType inst:IfcPump_364 ;
    ifc:relatingType_IfcRelDefinesByType inst:IfcPumpType_1354 .

```

Lis. 22 Ausgewählte Triple aus XYZ_ifc.ttl gemäß Tab. 46

Tab. 42 Erläuterung der Klassen im Alignment

Erläuterung	Beispiel des Lis. 22
Die Klasse <i>aimaix:IfcPumpType_CIRCULATOR</i> umfasst alle <i>ifc:IfcPumpType</i> -Objekte, deren Attribut <i>predefinedType CIRCULATOR</i> lautet.	<i>inst:IfcPumpType_1354</i>
Die Klasse <i>aimaix:IfcRelDefinesByType_P_C</i> beschreibt eine <i>RelDefinedByType</i> -Relation, deren Attribut <i>relatingType</i> der <i>aimaix:IfcPumpType_CIRCULATOR</i> ist	<i>inst:IfcRelDefinesByType_1410</i>
Die Klasse <i>aimaix:P_C</i> umfasst alle Objekte, die eine Typisierungsrelation der Klasse <i>:IfcRelDefinesByType_P_C</i> aufweisen	<i>inst:IfcPump_364</i>

4.3.4.2 Reasoning

Das Reasoning muss auf der gesamten Knowledge-Base erfolgenden, d.h. es müssen zuvor die verschiedenen KG (siehe Abb. 35) zusammengeführt werden. Im Falle einer Übersetzung von IFC nach Modelica (UC1) sind dies:

- xyz_ifc.ttl (siehe Abschnitt 4.3.3)
- IFC-OWL (2.1.2)
- Sprach-Ontologien MoOnt (3.6) und Express (2.1.2),
- KG der verwendeten Modelica-Bibliotheken AixLib (3.7) und
- Alignment IFC-AixLib (4.3.4.1).

Im Falle einer umgekehrten Übersetzung von Modelica nach IFC (UC2) sind dies

- xyz_mo.ttl,

⁴² https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/alignments/AlMAix_0.5.0.ttl

- Library-Ontologien, aller im Beispiel verwendeten Modelica-Bibliotheken,
- Sprach-Ontologien MoOnt und Express,
- IFC-OWL und
- Alignment IFC-AixLib.

Die Implementierung der wesentlichen Schritte erfolgte mit der Python-Bibliothek *OWLReady2*, außerdem kam *RDFLib* zum Einsatz. Für die Verwaltung der Wissensgraphen wurde die in *RDFLib*-integrierten Engine verwendet. Die gebildete Knowledge-Base wird im Arbeitsspeicher gehalten, ein Triplestore kam nicht zum Einsatz. Die Persistierung erfolgte mit Dateien, die als Turtle und RDF/OWL serialisiert wurden.

Das Reasoning erfolgt mit Hilfe des Reasoners *Pellet*, welcher in der *OWLReady-Python*-Bibliothek (siehe Tab. 54) zur Verfügung steht. Die geschlussfolgerten Tripel werden, wie in Lis. 23 farbig hervorgehoben, in einen neuen Wissengraph namens *moinst_python* eingefügt. Dieser KG entspricht dem ZS *XYZ_classes_mo.ttl* gemäß Abb. 39. Abweichend von der Begriffsverwendung in dieser Arbeit, wird dieser KG in der *OWLReady*-Bibliothek als *ontology* bezeichnet.

```
generate_combined_graph_file(filepaths_models_ttl, path, filename_original)
with moinst_python:    sync_reasoner_pellet(infer_property_values = True)
with moinst_python:
    insert_triples_header(default_world, sparql_header)
    insert_triples_components(default_world, sparql_header)
    insert_triples_components_placement(default_world, sparql_header)
    insert_triples_components_class_assignment(default_world, sparql_header)
    insert_triples_port_description(default_world, sparql_header)
    insert_triples_port_connections(default_world, sparql_header)
```

Lis. 23 Semantic Reasoning⁴³

4.3.4.3 Enhance_mo

Wie bereits in Abschnitt 3.8.2 thematisiert, ist es nicht möglich die Inhalte von *DataTypeProperties* durch Reasoning zuzuordnen. Weiterhin unterstützen die Reasoner auch keine *PropertyChains* variabler Länge. Die Zuordnung von *DataTypeProperties* ist jedoch notwendig, um die erzeugten Modelica-Modelle mit Parametern und Meta-Informationen wie folgt zu versehen:

- GlobalId als *stringComment* der Komponente
- Attribut *Name* des *IfcProduct* als *identifier* der Modelica-Komponente
- *IfcProperty* zur Lage der Komponente in der 2D-Darstellung (siehe Ausführungen im Abschnitt 3.5.4)
- Bezeichnung des ifc-seitigen Quell-Wissensgraph als *stringComment* des Modelica-Modells

Es kommen für deren Übertragung *SPARQL-INSERT*- und *SPARQL-SELECT*-Queries zum Einsatz, wie die folgenden Codeausschnitte Lis. 24 bis Lis. 26 zeigen. Diese ergänzen die entsprechenden Triple im Wissensgraph *XYZ_classes_moont.ttl* gemäß Abb. 39. Die Implementierung erfolgte als Python-Funktionen unter Nutzung der *RDFLib* im Modul *TranslationHelper*⁴⁴.

⁴³ https://github.com/ElisEck/MO-x-IFC/blob/main/04_SemTran/04_SemanticTranslator.py

⁴⁴ <https://github.com/ElisEck/MO-x-IFC/blob/main/PythonLib/TranslationHelper.py>

```

INSERT {
  ?obj a moont:MComponent.
  ?obj moont:identifier ?namev.
  ?obj moont:comment ?guidv.
} WHERE {
  ?obj ifc:globalId_IfcRoot / express:hasString ?guidv.
  ?obj ifc:name_IfcRoot / express:hasString ?namev.
  ?obj rdf:type / rdfs:subClassOf* moon:MComponent.
} """")

```

Lis. 24 SPARQL query für die Ergänzung von Informationen im Mo-KG⁴³

Namen von Modelica-Komponenten dürfen nur einen sehr eingeschränkten Zeichenumfang haben, dieser ist beim Attribut Name der Klasse `IfcRoot` weniger restriktiv. Es wird daher ein Postprocessing vorgenommen, um ggf. nicht erlaubte Zeichen durch Underlines zu ersetzen. Die Zuordnung zum Originalobjekt in IFC wird dadurch nicht beeinträchtigt, da die GlobalID des `IfcProduct` als `stringComment` zusätzlich übertragen wird.

Für die Übertragung der Position einer Komponenten wird auf Basis des Ergebnisses einer SPARQL SELECT query aus zwei einzelnen IfcProperties, ein String zusammengebaut (Lis. 25).

```

result = list(
    default_world.sparql(
        sparql_header + """
        SELECT ?obj ?namev ?guidv ?x_coord ?y_coord {
            ?irdbp ifc:relatedObjects_IfcRelDefinesByProperties ?obj.
            ?irdbp ifc:relatingPropertyDefinition_IfcRelDefinesByProperties ?ps.
            ?obj ifc:globalId_IfcRoot ?guid.
            ?guid express:hasString ?guidv.
            ?obj ifc:name_IfcRoot ?name.
            ?name express:hasString ?namev.
            ?ps ifc:hasProperties_IfcPropertySet ?psvx.
            ?psvx a aimaix:IfcPropertySingleValue_x_coord .
            ?psvx ifc:nominalValue_IfcPropertySingleValue ?r.
            ?r express:hasDouble ?x_coord.
            ?ps ifc:hasProperties_IfcPropertySet ?psvy.
            ?psvy a aimaix:IfcPropertySingleValue_y_coord .
            ?psvy ifc:nominalValue_IfcPropertySingleValue ?r2.
            ?r2 express:hasDouble ?y_coord.
        }""")
for res in result:
    coord_string = "\\" + str(res[3]) + ", " + str(res[4]) + "\\"
    new_axiom = "<" + res[0].namespace.base_iri + res[0].get_name() + ">
        moont:placement_origin " + coord_string + "."
    default_world.sparql(
        sparql_header + """
        INSERT {"""
        + new_axiom +
        """} WHERE{} """
        , error_on_undefined_entities=False)

```

Lis. 25 Codeausschnitt zur Übertragung der Positionsinformationen aus IFC-KG in Mo-KG⁴³

Da auch keine PropertyChains beim Reasoning unterstützt werden, ist für die Übertragung der Verbindungen zwischen den Komponenten eine weitere SPARQL-INSERT-Query nötig – siehe Lis. 26. Man erkennt, dass die Port-Bezeichnungen in IFC verwendet werden, um auch Modelica-seitig zuzuordnen welcher Port einer Komponente mit welchem Port einer anderen Komponente verbunden ist. Dabei kommt es nicht darauf an, dass dies exakt der avisierten Strömungsrichtungen entspricht. Es wird damit lediglich sichergestellt, dass es keine mehrfachen Verbindungen ausgehend von ein- und demselben Modelica-Port gibt, denn in diesem Falle würde die dahinterliegende Modelica-Komponenten „kurzgeschlossen“, d.h. in der Berechnung nicht berücksichtigt. Werden falsche Ports verbunden kommt es gegebenenfalls zu negativen Volumenströmen, d.h. Volumenströmen, die andersherum als erwartet, fließen. Die Modelica-Komponenten können damit jedoch i.d.R. umgehen (siehe dazu die Ausführungen zur aksualen

Modellierung in Abschnitt 2.2.1). Die negativen Volumenströme können zu erhöhten Rechenzeiten und/oder Problemen bei der Initialisierung der Modelle führen. Solche „Fehler“ können in Modelica allerdings nach der Berechnung leicht erkannt und durch Drehung der Komponenten behoben werden, dafür wurde keine Automatisierung vorgesehen.

```
INSERT {
    ?port1 moont:connectedTo ?port2.
    ?obj1 moont:hasPart ?port1.
    ?obj2 moont:hasPart ?port2.
    ?port1 a moont:MConnectorComponent.
    ?port2 a moont:MConnectorComponent.
} WHERE {
    ?port1 moont:connectedTo ?port2.
    ?obj1 moont:hasPart ?port1.
    ?obj2 moont:hasPart ?port2.
    ?port1 a ifc:IfcDistributionPort.
    ?port1 ifc:description_IfcRoot / express:hasString ?descr1.
    ?port2 a ifc:IfcDistributionPort.
    ?port2 ifc:description_IfcRoot / express:hasString ?descr2.
}
```

Lis. 26 Codeausschnitt zur Übertragung der connections aus IFC-KG in Mo-KG⁴³

4.3.5 TTL2MO Erzeugung von Modelica aus ttl (Konverter 5)

Als letzter Schritt im BIM2SIM-UC ist eine Transkription vom MoOnt-basierten Wissensgraph in ein natives Modelica-File notwendig. Dieses wurde in Python mit Hilfe der Libraries *RDFLib* und *modelica_builder* (siehe Tab. 54) implementiert.

Für das Einlesen des KG *XYZ_mo.ttl* kommt die *RDFLib* zum Einsatz. Lis. 27 und Lis. 28 zeigen den Quelltext. Zunächst werden mittels SPARQL-Queries die wesentlichen Informationen zu Komponenten und ihren Ports aus dem Wissensgraph extrahiert. Anschließend werden daraus die zugehörigen Komponenten in Modelica erzeugt und grob parametriert. Für jede Komponente wird die Klasse, der Modelica identifier, sowie die GlobalId und die Platzierung aus dem Wissensgraph übernommen. Weiterhin wird aus dem KG entnommen, welche Komponenten, mit Hilfe welcher Ports verbunden sind.

```

from ModelicaHelper import modelica_name_from_ifc_name, add_model_to_package, add_connection, in-
    sert_component_at_location, add_connection_2, insert_component_at_location_2

q_components = """
PREFIX moont: <https://w3id.org/moont/>
select DISTINCT ?comp ?ident ?guid ?placem ?class where {
    ?comp a moont:MComponent.
    ?comp moont:identifier ?ident.
    ?comp moont:comment ?guid.
    ?comp moont:placement_origin ?placem.
    ?comp a ?class.
    FILTER(STRSTARTS(STR(?class), "https://w3id.org/aix/")) .
}
"""

componentlist = list(GA.query(q_components))

set_x = set()
set_y = set()
for component in componentlist:
    lower_left_corner = component[3].value[1:-1].rsplit(', ', ) #[x, y]
    set_x.add(float(lower_left_corner[0]))
    set_y.add(float(lower_left_corner[1]))
min_x = min(set_x)
min_y = min(set_y)
dict_old_to_new_placement = {}
for component in componentlist:
    old_placement = component[3].value
    lower_left_corner_old = component[3].value[1:-1].rsplit(', ')
    lower_left_corner_new = [int(float(lower_left_corner_old[0])-min_x), int(float(lower_left_corner_old[1])-min_y)]
    dict_old_to_new_placement[old_placement] = lower_left_corner_new

for component in componentlist:
    identifier = component[1].value.replace("(", "_").replace(")", "_")
    string_comment = component[0].rsplit('/')[-1] + " - " + component[2].value
    lower_left_corner = dict_old_to_new_placement[component[3].value]
    modelica_class = component[4].rsplit('https://w3id.org/aix/')[1]
    modifications = {'Medium': 'Medium', 'm_flow_nominal': 'mp_nom', 'dp_nominal': 'dp_nom'}
    model = insert_component_at_location_2(model, modelica_class, identifier, string_comment,
                                           lower_left_corner, modifications)

```

Lis. 27 TTL2MO Teil 1: Komponenten finden und erzeugen (Parametrierung in orange)⁴⁵

Abb. 62 auf S. 159 zeigt den Diagram-Layer des erzeugten Modells. Die Größen, Abstände und Schriftgrößen wurden so gewählt, dass die Erkenn- und Handhabbarkeit auf handelsüblichen Bildschirmgrößen gut ist. Es ist die grundsätzliche Struktur aus Abb. 61 gut wieder zu erkennen.

Tab. 43 listet die Parameter der im Rahmen dieser Arbeit verwendeten AixLib-Bibliotheks-Komponenten auf. Man erkennt, dass i.d.R. eine Angabe zum enthaltenen Medium der Komponenten (dunkelgrüner Text; übliche Werte: Wasser, Wasser-Glycol-Gemisch-30%,), zum Nennmassestrom (orange) und zum Nenndruckverlust (hellgrün) nötig sind. Da Informationen dazu i.d.R. im IFC-Modell nicht enthalten sein werden, werden bei allen Komponenten die entsprechenden Parameter mit einem globalen Parameter des Simulationsmodells belegt. Damit kann der Simulationsanwender diese Werte an einer zentralen Stelle einmalig bestimmen und erhält so schnellstmöglich ein rechenfähiges Modell (siehe oranger Text in Lis. 27). Komponenten mit signifikanten Druckverlusten müssen anschließend manuell nachparametriert werden. Von einer Standardbelegung dieser Parameter wurde bewusst abgesehen, um eine korrekte Größenordnung vom Nutzer abzufragen.

⁴⁵ https://github.com/ElisEck/MO-x-IFC/blob/main/05_TTL2MO/05_TTL2MO.py

```

query_ports = """
PREFIX moont: <https://w3id.org/moont/>
select DISTINCT ?oid1 ?pid1 ?pl1 ?oid2 ?pid2 ?pl2 where {
    ?port1 moont:connectedTo ?port2.
    ?obj1 moont:hasPart ?port1.
    ?obj2 moont:hasPart ?port2.
    ?port1 moont:identifier ?pid1.
    ?port2 moont:identifier ?pid2.
    ?obj1 moont:identifier ?oid1.
    ?obj2 moont:identifier ?oid2.
    ?obj1 moont:placement_origin ?pl1.
    ?obj2 moont:placement_origin ?pl2.
}
"""

connectionlist_rdflib = list(GA.query(query_ports))
for con in connectionlist_rdflib:
    component1 = con[0].value.replace("(", "_").replace(")", "_")
    component1_portname = con[1].value.replace("(", "_").replace(")", "_")
    center_1 = dict_old_to_new_placement[con[2].value]
    component2 = con[3].value.replace("(", "_").replace(")", "_")
    component2_portname = con[4].value.replace("(", "_").replace(")", "_")
    center_2 = dict_old_to_new_placement[con[5].value]
    model = add_connection_2(model, component1, component1_portname, component2,
                             component2_portname, center_1, center_2)

add_model_to_package(model, model_name, package_name, package_path)

```

Lis. 28 TTL2MO Teil 2: Finden und Erzeugen von Verbindungen⁴³

Tab. 43 Parameter der verwendeten AixLib Komponenten

AixLib-Komponente	Parameter
AixLib.Fluid.FixedResistances.LosslessPipe	Medium M_flow_nominal
AixLib.Fluid.HeatExchangers.SensibleCooler_T	Medium Qmin_flow M_flow_nominal Dp_nominal
AixLib.Fluid.Sensors.MassFlowRate	Medium
AixLib.Fluid.HeatExchangers.ConstantEffectiveness	Medium1 Medium2 M1_flow_nominal M2_flow_nominal Dp1_nominal Dp2_nominal
AixLib.Fluid.FixedResistances.Junction	Medium M_flow_nominal Dp_nominal
AixLib.Fluid.Movers.FlowControlled_m_flow	Medium (M_flow_nominal Dp_nominal) – haben Standardbelegung
AixLib.Fluid.FixedResistances.CheckValve	Medium m_flow_nominal dp_Valve_nominal
AixLib.Fluid.Actuators.Valves.TwoWayLinear	
AixLib.Fluid.Actuators.Valves.ThreeWayLinear	

4.3.6 MoTTL-Transcriptor (Konverter 7)

Für die Übertragung eines nativen Modelica-Files in einen Wissensgraphen kommt der MoTTL-Transcriptor zum Einsatz. Dieser wurde bereits in Abschnitt 3.7 im Kontext der Erstellung der Library-Ontologien beschrieben und kommt auch für die Transkription von *Modelica-Executables* zum Einsatz.

Der *MoTTL-Transcriptor* wurde in Java implementiert. Zunächst wurde mit dem *ANTLR* ein Modelica-Parser-Rahmenwerk erstellt. *ANTLR* (ANother Tool for Language Recognition) (siehe Tab. 54) ist ein leistungsfähiger Parser-Generator zum Lesen, Verarbeiten, Ausführen oder Übersetzen

von strukturierten Text- oder Binärdateien. *ANTLR* benötigt eine Grammatikdatei, in der die zu parsende Sprache formal spezifiziert ist. Eine solche Grammatikdatei in BNF-Form findet sich im Anhang der Modelica Language Specification⁴⁶ (siehe Tab. 54). Die Grammatikdatei enthält Informationen welche Schlüsselwörter und Trennzeichen verwendet werden und welche Reihenfolge für die Interpretation der Informationen in der zu parsenden Datei relevant ist. Auf der Grundlage der Grammatik generiert *ANTLR* einen *Parser-Stub*, der verwendet werden kann, um den durch die Modelica-Datei repräsentierten Baum zu durchlaufen. Der generierte *Parser-Stub* besteht aus mehreren Java-Klassen und -Schnittstellen, die anschließend implementiert wurden, um die notwendige Funktionalität zur Transformation der Modelica-Datei in einem internen Datenmodell bereitzustellen. Für das interne Datenmodell wurde eine Methode *serializeToTTL* implementiert, die das interne Datenmodell als Graph in Turtle-Syntax serialisiert.

Der MoTTL-Transcriptor ist als Kommandozeilen-Tool verfügbar⁴⁷, sein wichtigster Eingabeparameter ist der Pfad zu einem Modelica-Paket. Als Ausgabe erzeugt der Transkriptor einen Wissensgraphen dieses Pakets in Form einer Turtle-Datei.

Der MoTTL-Transcriptor wurde entsprechend den MoOnt-Versionen zwei Versionen veröffentlicht v0.9.0 und v1.0.0 (siehe Tab. 44). Letztere enthält zusätzlich die Transkription des *placement_string*, welche für den SIM2BIM-UseCase notwendig ist.

Tab. 44 zusammengehörige Versionen von MoOnt und MoTTL-Transcriptor

MoOnt	MoTTL-Transcriptor	
0.9.2 (23.6.23)	0.9.0	https://github.com/ElisEck/MO-x-IFC/releases/tag/v0.9.0 (23.6.23)
1.0.0 (27.10.23)	1.1.0	https://github.com/ElisEck/MO-x-IFC/releases/tag/v1.1.0 (27.2. bzw. 16.8.23)

4.3.7 ProTran Procedural Translator (Konverter 8)

Den Ausführungen in Abschnitt 4.3.4.2 war bereits zu entnehmen, dass das Reasoning zwar einen Großteil der notwendigen Informationen für die Erstellung eines Modelica-Modells aus IFC (und umgekehrt) bereitstellen kann, jedoch ein erhebliches Postprocessing mit SPARQL-queries (4.3.4.3) notwendig ist.

Es wurde daher auch ein Lösungsansatz, ausschließlich mit SPARQL-queries geprüft. Dessen wesentliche Inhalte sind in Lis. 29 dargestellt. Farbig hervorgehoben ist die Funktion, welche das in Abschnitt 4.3.4.2 beschriebene Reasoning ersetzt. Diese ist im Lis. 30 detailliert wiedergegeben. Die Erstellung eines Alignment (4.3.4.1) als Wissensgraph kann ebenfalls entfallen. Man erkennt, dass die Mapping-Tabelle (Tab. 38) in diesem Fall nicht in Form eines Alignments in die Übersetzung einfließt, sondern direkt Bestandteil des Quelltextes wird.

```
graph_mo = Graph()
graph_mo.parse(input_path + shortname + '_moont.ttl', format='turtle') #0.2MB

graph_ifcowl = generate_instance_objects(graph_mo, lineno=1)
graph_ifcowl = add_metadata_to_graph(graph_ifcowl, input_path + shortname + '_moont.ttl')
graph_ifcowl = add_placement_properties_to_objects(graph_mo, lineno + 1, graph_ifcowl)
graph_ifcowl = generate_IfcRelConnectsPorts(graph_mo, lineno)
graph_ifcowl = generate_IfcRelNests_for_ports(graph_mo, lineno)
graph_ifcowl = add_modelica_class_and_stringComment_as_property_to_objects(graph_mo, lineno + 1)
```

Lis. 29 ProTran: wesentliche Bestandteile (nur ausschnittsweise wiedergegeben)⁴⁸

⁴⁶ <https://github.com/antlr/grammars-v4/blob/master/modelica/modelica.g4>

⁴⁷ <https://github.com/ElisEck/MO-x-IFC/releases/>

⁴⁸ https://github.com/ElisEck/MO-x-IFC/blob/main/08_ProTran/08_ProceduralTranslator.py

4 Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web

4.3 Details zu den einzelnen Konvertern MOXIK

4.3.7 ProTran Procedural Translator (Konverter 8)

```

def generate_instance_objects(moont_graph, lineno):
    dict_mapping = {
        'IfcPipeSegment_USERDEFINED': ['aix:AixLib.Fluid.FixedResistances.LosslessPipe'],
        'IfcPipeFitting_JUNCTION': ['aix:AixLib.Fluid.FixedResistances.Junction'],
        'IfcValve_CHECK': ['aix:AixLib.Fluid.FixedResistances.CheckValve'],
        'IfcValve_ISOLATING': ['aix:AixLib.Fluid.Actuators.Valves.TwoWayLinear'],
        'IfcValve_MIXING': ['aix:AixLib.Fluid.Actuators.Valves.ThreeWayLinear'],
        'IfcHeatExchanger_USERDEFINED': ['aix:AixLib.Fluid.HeatExchangers.HeaterCooler_u',
            'aix:AixLib.Fluid.HeatExchangers.ConstantEffectiveness'],
        'IfcSensor_FLOWSENSOR': ['aix:AixLib.Fluid.Sensors.EnthalpyFlowRate'],
        'IfcSensor_HEATSENSOR': ['libeas:LibEAS.Sensors.WMZ'],
        'IfcSensor_PRESSURESENSOR': ['aix:AixLib.Fluid.Sensors.Pressure'],
        'IfcSensor_TEMPERATURESENSOR': ['aix:AixLib.Fluid.Sensors.TemperatureTwoPort'],
        'IfcPump_CIRCULATOR': ['aix:AixLib.Fluid.Movers.SpeedControlled_y'],
        'IfcTank_EXPANSION': ['aix:AixLib.Fluid.Sources.Boundary_pT'],
        'IfcTank_STORAGE': ['aix:AixLib.Fluid.Storage.Stratified']
    }
    set_of_translated_modelica_idents = set()
    set_of_translated_modelica_classes = set()

    ifcowl_graph = Graph()
    for ifc_class_type in dict_mapping:
        for modelica_class in dict_mapping[ifc_class_type]:
            result_all_comps = query_for_modelica_components_of_certain_class(moont_graph, modelica_class)
            for line in result_all_comps:
                comp = line.get('comp')
                ident = line.get('ident')
                set_of_translated_modelica_idents.add(ident)
                set_of_translated_modelica_classes.add(modelica_class)
                triples = construct_ifc_instance(moont_graph, comp, ifc_class_type, lineno, ident)
                lineno += 2
                ifcowl_graph = ifcowl_graph + triples.graph
    return ifcowl_graph, lineno

def query_for_modelica_components_of_certain_class(moont_graph, modelica_class):
    query_comps = """
        SELECT DISTINCT ?comp ?ident
        WHERE {
            hil:HIL.HIL_flat moont:hasPart ?comp .
            ?comp rdf:type / moont:extends* """ + modelica_class + """ .
            ?comp moont:identifier ?ident .
        } ORDER BY ASC(?comp)
    """
    # GUIDs vergeben (funktioniert nicht in einer construct query)
    result_all_comps = moont_graph.query(query_comps)
    return result_all_comps

```

Lis. 30 ProTran: Funktionen zur Erzeugung der Klassenzuordnungen (Ersatz für das Reasoning)⁴⁹

Ebenso, wie bei der Anwendung des Reasoning ist auch in bei der Procedural Translation anschließend ein *Enhancement* des Rohgraphen (*XYZ_classes.ttl*) notwendig. Wie Lis. 29 zeigt, gehören dazu z.B. die Erzeugung der Ports und Verbindungen, sowie die Ergänzung der *IfcProperties*.

Für die Übersetzung müssen neben dem Instanzgraphen des Modelica-Modells (Mo-KG), auch die Library-Graphen verarbeitet werden (*MSL*, *Aix*, *LibEAS*), um festzustellen welcher Modelica-Klasse die Komponenten eines Modells zugehörig sind. Die relevanten Library-Ontologien werden daher in das Instanzmodell importiert, wie Lis. 36 zeigt.

Insbesondere wird dieser Import relevant, wenn das Mapping mit Hilfe von Klassen erfolgt, welche im Modell nicht direkt verwendet werden, sondern stattdessen *Kind-Klassen* davon. Solange benutzerdefinierte Bibliotheks-Komponenten von einer im Mapping enthalten Klasse (in beliebiger

⁴⁹ https://github.com/ElisEck/MO-x-IFC/tree/main/08_ProTran/HelperProceduralTranslation.py

Generation) erben (Schlüsselwort *extends* in Modelica), werden sie vom *Procedural Translator* automatisch mit behandelt (Beispiel in Lis. 31). Ausschlaggebend ist die in Lis. 30 farbig hervorgehobene *PropertyChain*.

```
@prefix moont: <http://w3id.org/moont#> .
@prefix aix: <http://www.eas.iis.fraunhofer.de/aix#> .
@prefix libeas: <http://www.eas.iis.fraunhofer.de/libeas#> .

libeas:LibEAS.Sensors.PressureDrop_Display moont:extends aix:AixLib.Fluid.FixedResistances.LosslessPipe.
```

Lis. 31 Ausschnitt aus der Library-Ontologie einer benutzerdefinierten Modelica-Bibliothek

4.3.8 RDF2IFC Überführung IFC-OWL in IFC-STEP (Konverter 9)

Für die anschließende Überführung von IFC-OWL-Files (*XYZ_ifc.ttl*) in IFC-STEP-Dateien wird das Tool IfcOWL-to-IfcSTEP verwendet, welches von (Zhang 2019) basierend auf den Vorarbeiten von Pauwels in Java implementiert wurde. Es steht unter Apache License 2.0 frei zur Verfügung.

Die Implementierung des Konverters wurde geringfügig angepasst, da die Ausgabe von IFCREAL-Entitäten im Tool von (Zhang 2019) nicht IFC-Schema-konform erfolgte. Weiterhin wurde die URL des IFC-4-Namespace angepasst, da sie veraltet war. Der derart angepasste Konverter wurde als Version 1.1.0 neu kompiliert und veröffentlicht.

Lis. 32 zeigt den Aufruf, es sind lediglich Quell- und Zielfile als Eingabeparameter anzugeben.

```
java -jar IfcOWL2IfcSTEP.jar <input.xxx> <output.ifc> [options]
```

Lis. 32 Aufruf der IfcOWL-to-IfcSTEP Kommandozeilenanwendung

4.3.9 Voluminizer: Erzeugen von Platzhaltergeometrien in IFC (Konverter 10)

Bei der Transkription aus IFC-OWL nach IFC-STEP werden lediglich vorhandene Informationen übertragen. Informationen zur Geometrie der IFC-Komponenten sind in Modelica nicht enthalten. Für das Handling in IFC- und CAD-Autorentools ist es jedoch i.d.R. notwendig eine dreidimensionale Repräsentation zu vorzufinden, um die Objekte manipulieren zu können. Es wird daher bei Objekten, die erstmals aus Modelica-Modellen erzeugt werden, eine Platzhaltergeometrie erzeugt. Die Lageinformation wird dabei aus der Position im Modelica-Diagramm entnommen. Die Geometrie wird entsprechend der Klassifizierung und Typisierung in Tab. 45 erzeugt.

Die Farben wurden aus einer Palette für kategorische Variablen gewählt, um eine gute Unterscheidbarkeit zu gewährleisten. Dabei wurden die Standardfarben des KIT-Viewer (pink, türkis) außen vor gelassen.

4 Implementierung eines Toolsets für die bidirektionale Übersetzung mittels Semantic Web

4.3 Details zu den einzelnen Konvertern MOXIK

4.3.9 Voluminizer: Erzeugen von Platzhaltergeometrien in IFC (Konverter 10)

Tab. 45 Platzhalter-Geometrien

Klasse	Form**	Größe (Skalierung)	Farbe*
IfcAirTerminal	Quader	0.5	Rot
IfcAirTerminalBox	Quader	0.5	Rosa
ifcAirtoAirHeatRecovery	Quader	1	Rot
IfcCoil	Quader	1	Lila
IfcDamper	Zylinder liegend	0.5	Orange
IfcDistributionElement	Quader	1	Orange
ifcDuctSilencer	Zylinder liegend	0.5	Grün
ifcFan	Quader Kugel	1.5	Türkis
ifcFilter	Zylinder liegend	0.5	Blau
IfcFlowInstrument	Zylinder liegend	0.5	Rosa
IfcHeatExchanger	Quader	1	Grün
IfcPipeFitting	Zylinder liegend	0.5	Grau
IfcPipeFitting	Zylinder liegend	2	Grau
IfcPipeSegment	Zylinder liegend	0.2	Grau
IfcPump	Quader Kugel	1.5	Blau
IfcSensor	Zylinder liegend	0.5	Rot
IfcTank	Zylinder stehend	2	Keine
IfcValve	Zylinder liegend	0.5	Lila

* zur Anzeige im KIT Viewer (Display-Colours-From Entity)

** KIT-Viewer kann keine Kugeln, daher Quader

Analog zur Beschreibung auf S. 131 wird zusätzlich eine 2D-Repräsentation für alle Objekte hinzugefügt.

5 Experimente

In Kapitel 1 wurden die zwei UseCases BIM2SIM und SIM2BIM beschrieben. Beide UseCases wurden in diesem Kapitel experimentell mittels der Software-Komponenten aus Kapitel 4 umgesetzt.

Die Umsetzung des BIM2SIM-UseCase erfolgt am Beispiel des Kältesystems eines Institutsgebäudes. Das dafür existierende BIM-Modell wurde in ein simulationsfähiges Modelica-Modell überführt. Die Herstellung der Voraussetzungen und die notwendigen Nacharbeiten werden dargestellt.

Die Umsetzung des SIM2BIM-UseCase erfolgt am Beispiel eines Versuchsstandes, welcher innerhalb der Simulationsumgebung Dymola entworfen wurde. Dieser wird in ein valides IFC-Modell mit einer Platzhaltergeometrie überführt, welches anschließend in Autorentools weiter bearbeitet werden kann.

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.1 Überblick

5.1 UseCase 1: BIM2SIM

5.1.1 Überblick

Der UseCase BIM2SIM hat seine praktische Anwendung in der Nachrechnung bereits fortgeschritten geplanter oder sogar realisierter Anlagen. Voraussetzung ist die Existenz eines Konstruktionsmodells im IFC-STEP-Format (*XYZ.ifc* in Abb. 48). Entweder erfüllt dieses Modell bereits die Anforderungen aus Abschnitt 3.5.1 oder diese Voraussetzungen müssen zunächst hergestellt werden (*XYZ_K.ifc*). Anschließend werden die vier Konverter *Schematizer*, *IFC2OWL*, *SemTran* und *TTL2MO* angewendet, um eine Modelica-Datei mit valider Modellstruktur und geeigneter Vorparametrierung zu erstellen (*XYZ_raw.mo*). Abb. 48 gibt einen Überblick über die notwendigen Konverter aus dem MO-x-IFC-Toolset und die erzeugten Zwischenschritte für die Anwendung im BIM2SIM-UseCase. Die Anwendung der Konverter wird in diesem Abschnitt experimentell untersucht.

Es wurde dafür bewusst kein künstliches oder Minimal-Beispiel gewählt, um die Anwendbarkeit in der Praxis zu überprüfen. Um kurze Rechenzeiten (<10s) zu gewährleisten, wurde jedoch ein möglichst kleines System gewählt.

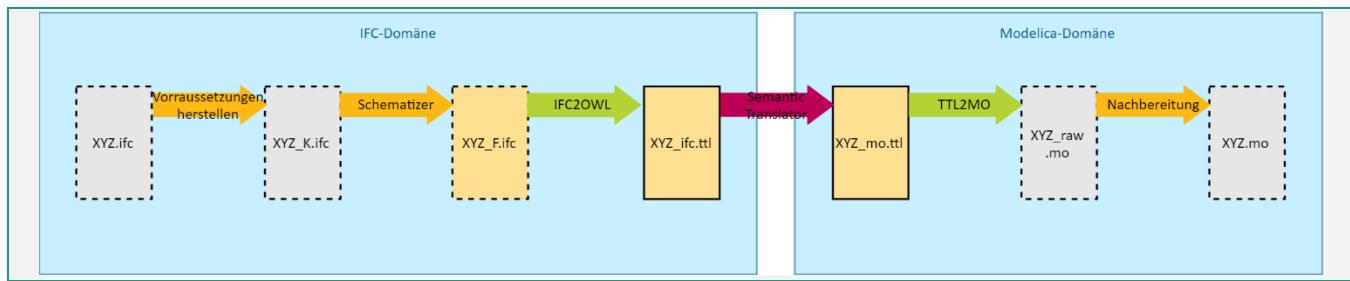


Abb. 48 Konverter für BIM2SIM

Tab. 46 Bezeichnung der Zustände im Experiment zu UC1 BIM2SIM

Allgemeine Bezeichnung	Pfad im digitalen Anhang Basispfad: https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/
XYZ_K.ifc	/Server_K.ifc
XYZ_F.ifc	/Server_F.ifc
XYZ_ifc.ttl	/Server_ifc.ttl
XYZ_mo.ttl	/Server_mo.ttl
XYZ_raw.mo	/GeneratedModels/generated.mo
XYZ.mo	

5.1.2 Demonstrator Serverkreis

Bei dem gewählten Beispiel handelt es sich um einen Kältekreis im Institutsneubau des Fraunhofer EAS in Dresden.

Einordnung in das Gebäude

Das Institutsgebäude ist ein zweiflügeliges Gebäude mit einer Nutzfläche von 4300m². Es bietet auf 3 bzw. 4 Etagen Platz für Büros, Labore und Besprechungsräume. Abb. 49 zeigt das IFC-Model der Architektur des Gebäudes.

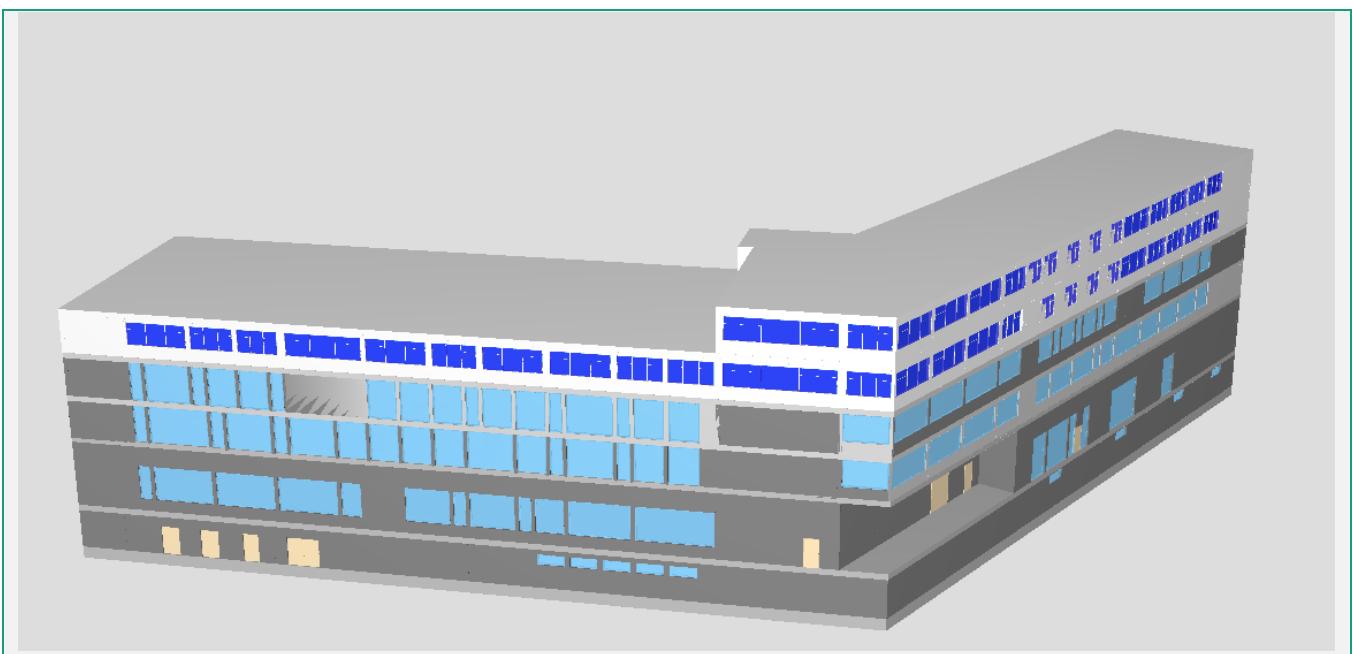


Abb. 49 IFC-Modell der Architektur des Beispielgebäudes

Bildquelle: Klemm, (Challagulla 2023) Visualisierung des Modells mit Simplebim 9.1

Das Gebäude enthält ein Kaltwassersystem zur Versorgung von Umluftkühlern, aktivierten Geschossdecken, Serverschränken und Lüftungsanlagen. Die Erzeugung erfolgt mittels einer wassergekühlten Kältemaschine bzw. freier Kühlung. Es stehen dafür zwei trockene Kühlertürme auf dem Dach des Gebäudes zur Verfügung. Abb. 50 zeigt einen Ausschnitt des Schaltschemas von Erzeugung und Rückkühlung des Kaltwassersystems, Abb. 51 die Geometrie des gesamten Kaltwassersystems.

Einordnung in das
Kaltwassersystem

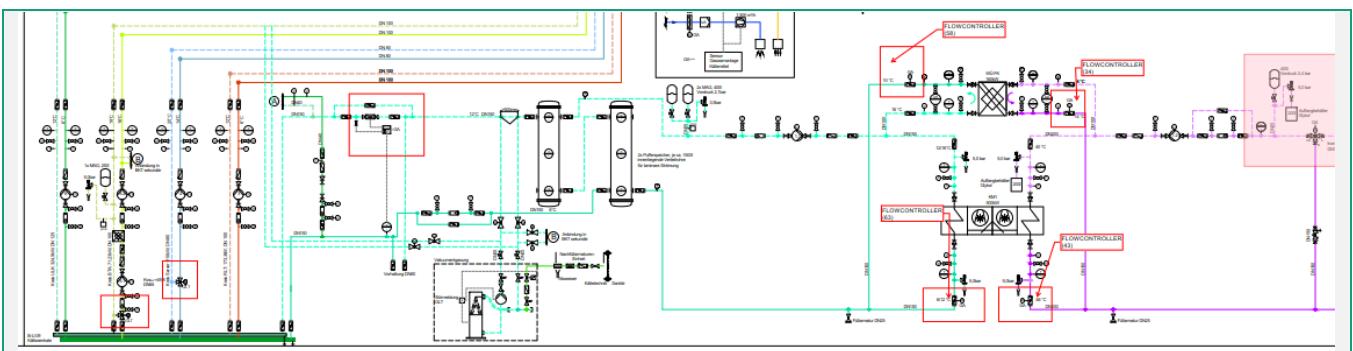


Abb. 50 Ausschnitt aus dem Schaltschema des Kaltwassersystems im Gebäude, vollständiges Schema in Abb. 47
(Quelle: Klemm Ingenieure)

Das Kaltwassersystem des Gebäudes besteht aus folgenden hydraulischen Subsystemen:

1. Erzeugung
2. Rückkühlung
3. Bauteilaktivierung
4. Server
5. Raumlufttechnik
6. Umluftkühler

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

Dabei sind die Nummern 3 bis 6 Kühlkreise. Der Serverkreis als kleinstes Subsystem soll im Folgenden als Beispiel diesen, er ist daher in Abb. 51 hervorgehoben.

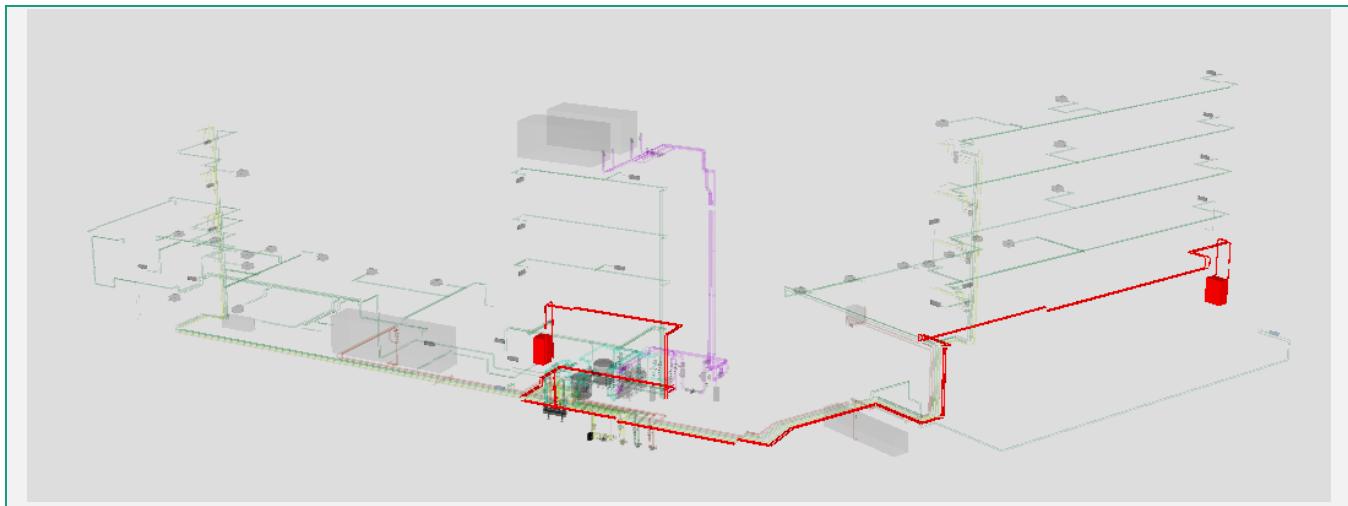


Abb. 51 Konstruktionsmodell des Kaltwassersystems (das betrachtete Teilsystem in rot)

(entspricht XYZ.ifc bzw. XYZ_K.ifc in Abb. 33 und Tab. 29)

Bildquelle: Klemm Ingenieure

Das gewählte Teilsystem Serverkreis versorgt zwei Serverschränke im Gebäude mit 14°C-Kaltwasser. Da die Erzeugung des Kaltwassers je nach Betriebsfall mit 6 bis 12°C erfolgt, handelt es sich um einen abgemischten Kaltwasserkreis, wie der Schemaausschnitt in Abb. 52 zeigt.

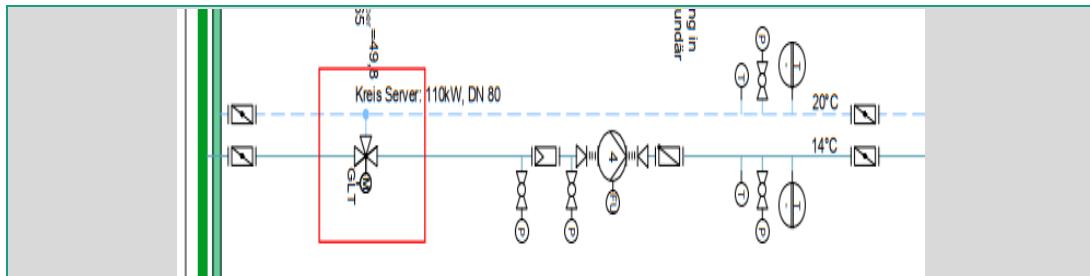


Abb. 52 Schemaausschnitt: Verteilerabgang Serverkreis (Quelle: Klemm)

5.1.3 Erprobung

5.1.3.1 Schritt 1: Vorbereitung

Der untersuchte UseCase ist sinnvoll nur auf abgeschlossene Systeme anwendbar. Diese zeichnen sich dadurch aus, dass sie keine bzw. möglichst wenige Schnittstellen zu anderen Teilsystemen haben. Derartige Schnittstellen müssen in der Simulation durch geeignete Randbedingungen ersetzt werden. Sinnvoll ist beispielsweise die Anwendung auf einzelne Heiz- und Kühlkreise, wie sie in Abschnitt 1.1.2 definiert wurden. Weniger sinnvoll wäre die Anwendung auf vereinzelte Räume oder Geschosse, da die zugehörigen Randbedingungen i.d.R. nicht bekannt sind. Der Workflow nutzt Dateien als kleinste Einheit. Es war daher zunächst zu gewährleisten, dass eine IFC-Datei bereitsteht, welche ein solches kleines System vollständig (und ausschließlich) enthält. Ungeeignet wären hingegen IFC-Dateien, welche einen räumlichen Ausschnitt von einem oder mehreren Systemen enthalten, wie z.B. häufig übliche geschoss- und bauabschnittsweise Dateien aller Heizungsrohre.

Ursprungsmodelle

Die für das Experiment verwendete Datei wurde aus Originaldateien aufbereitet, welche ohne gesonderte Anforderungen vom für die TGA-Planung beauftragten Planungsbüro mit der Software

AutoCAD MEP 2021 - Deutsch (German) Build 8.3.53.0 erstellt und anschließend geschossweise als IFC4 exportiert wurden. Diese Vorarbeiten wurden in (Urbanski 2021) beschrieben. Alle im folgenden beschriebenen Aufbereitungen erfolgten i.d.R. für das gesamte Kältemodell, daher sind im Folgenden auch andere Komponenten als jene aus dem Serverkreis thematisiert. Die verwendeten Originalmodelle waren geschossweise Exporte des Kältesystems des Gebäudes. Neben dem oben beschriebenen Kaltwassersystem waren hierin Splitsysteme sowie vereinzelte Lüftungsanäle enthalten, außerdem die Kondensatanschlüsse aller 43 Umluftkühler. Um geschlossene Systeme herzustellen, mussten 14 Abgänge für Betonkernaktivierung, welche im Rohbau-Modell ihre Fortsetzung finden, durch Platzhalter für Heizkreise (siehe Abschnitt 1.1.1.2) ersetzt werden.

Die in der Planung gebräuchlichen IFC-STEP-Dateien umfassen i.d.R. eine geometrische Darstellung der Anlagen, wie z.B. in Abb. 51 dargestellt. Gemäß des IFC-Schemas sind dafür viele IFC-Entitäten unterschiedlicher *Kind-Klassen* von *IfcProduct* notwendig. Das Beispielmodell in Abb. 51 enthält etwa 5200 Instanzen von *IfcProduct* (darunter 2400 *IfcFitting* und 2310 *IfcPipeSegment*). Diese sind mit Hilfe von *IfcDistributionPorts* miteinander verbunden und bilden einen Netzwerkgraphen mit Knoten und Kanten. Sie erfüllen damit die wichtigste Voraussetzung aus Abschnitt 3.5.1.

Abb. 53 zeigt eine Visualisierung des im Ursprungsmodell vorhandenen Anlagengraphen. Jeder Punkt stellt dabei ein *IfcProduct* dar, Verbindungen sind durch graue Linien dargestellt. Es wurde ein kraftbasiertes Layout angewendet (siehe Abschnitt 3.5.4.4). Die Farbe der Knoten zeigt die Systemzugehörigkeit an. Man erkennt, dass die allermeisten der *IfcProducts* Verbindungen zu zwei Nachbarobjekten haben, wie dies zu erwarten wäre. Innerhalb der meisten Systeme existieren mehrere Subgraphen, dies deutet auf fehlende Verbindungen hin. Das System besteht aus 42 Subgraphen. Der Sollzustand entspricht einem Graphen pro System, es müssen daher einige Verbindungen nachgepflegt werden. Aufgrund der hohen Anzahl fehlender Verbindungen wurde zunächst versucht, diese automatisiert anhand der Position der Elemente nachzupflegen. Dafür wurde eine Schwellenabstand definiert unterhalb dessen zwei benachbarte *IfcDistributionPorts* als verbunden betrachtet werden. Jedoch besitzen die *IfcDistributionPorts* im System keine eigene Geometrie und Position, weshalb hilfsweise die Daten des Objekts verwendet werden, welches die jeweiligen *IfcDistributionPorts* enthält.

Nachpflege von
Verbindungen

Um falsche Verbindungen zwischen parallel verlaufenden Rohrleitungen zu vermeiden, muss der Schwellwert kleiner sein als die in der Anlage verwendeten Leitungsabstände. Die kleinste regelmäßig verwendete Leitungsdimension (im Beispiel DN12) dient dafür als Anhaltspunkt.

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

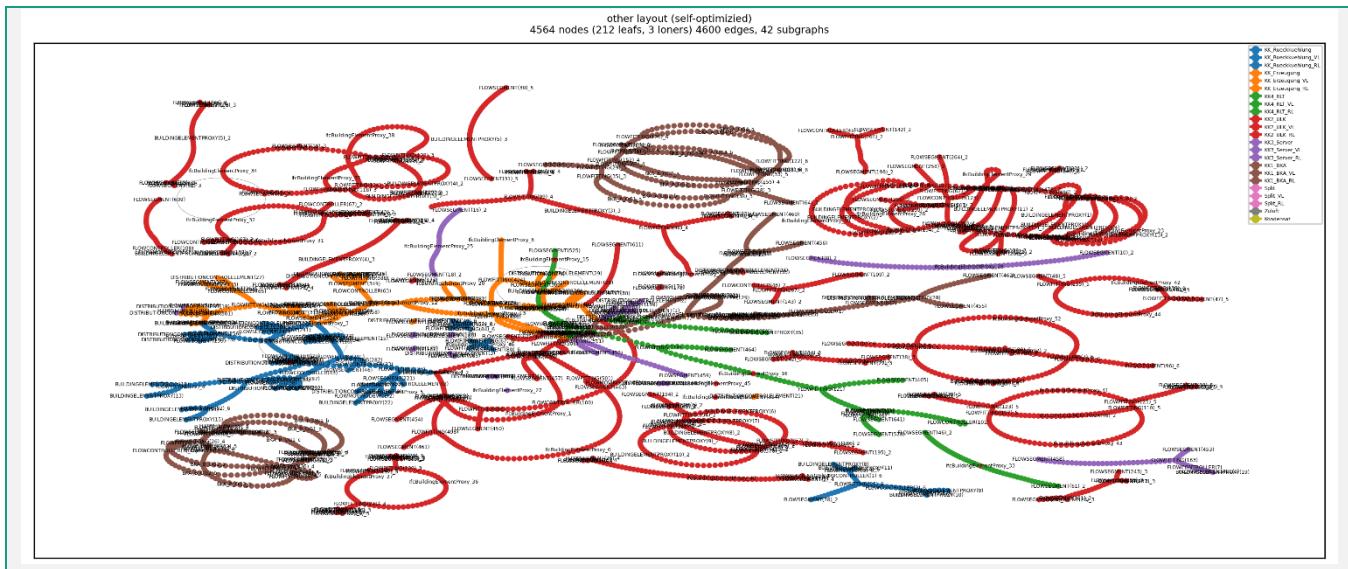


Abb. 53 Ursprungsgraph
(rot: ULK, braun: BKA, lila: Server, rosa: Split, grün: RLT, orange: Erzeugung, blau: Rückkühlung)

Abb. 54 zeigt rot hervorgehoben Elemente im Dachgeschoss, welche weniger als zwei Verbindungen zu benachbarten Elementen aufweisen. Dabei wird deutlich, dass die fehlenden Verbindungen an *IfcBuildingElementProxy*-Elementen bestehen. Deren Abmessungen sind üblicherweise relativ groß im Vergleich zu anderen Modellkomponenten wie z.B. Rohrstücken und Armaturen. Die automatische Erkennung der Nachbarschaftsbeziehung bedürfte daher einer sehr genauen Interpretation der Geometrie: Eine Auswertung lediglich eines Eckpunkts (z.B. Koordinatenursprung des Elementkoordinatensystems des betroffenen *IfcProduct*) und seiner Abstände zu anderen Eckpunkten ist insb. bei großen Komponenten wie den in Abb. 51 dargestellten Beispielen nicht erfolgversprechend, da der Schwellwert sehr groß eingestellt werden müsste, um ohne Kenntnis des konkreten behandelten Eckpunkts alle benachbarten Komponenteneckpunkte zu finden.

Es verbleibt kein sinnvoller Bereich zwischen der notwendigen Ober- und Untergrenze des Abstandsschwellwerts, so dass von einer geometriebasierten Zuweisung der Nachbarschaftsbeziehungen abgesehen werden musste.

Da aus den genannten Gründen eine automatische Zuordnung nicht machbar war, wurden die fehlenden Verbindungen manuell nachgepflegt: Dafür wurden die Anlagen(teil)graphen in Form von GraphML-Dateien in das Graphen-Visualisierungs- und -Bearbeitungstool *Gephi* importiert. Dort erfolgte die manuelle Verknüpfung aller offenen Enden des Graphen basierend auf den Bezeichnungen der *IfcProducts* und einer parallelen Sichtprüfung im IFC-Viewer. Der ergänzte Graph wurde als Tabelle bzw. GraphML-Datei exportiert und vom Nachpflegeskript (4.3.1.3) wieder eingelesen. Basierend darauf wurden die entsprechenden *IfcRelConnectsPorts*-Relationen im IFC-Modell nachgepflegt.

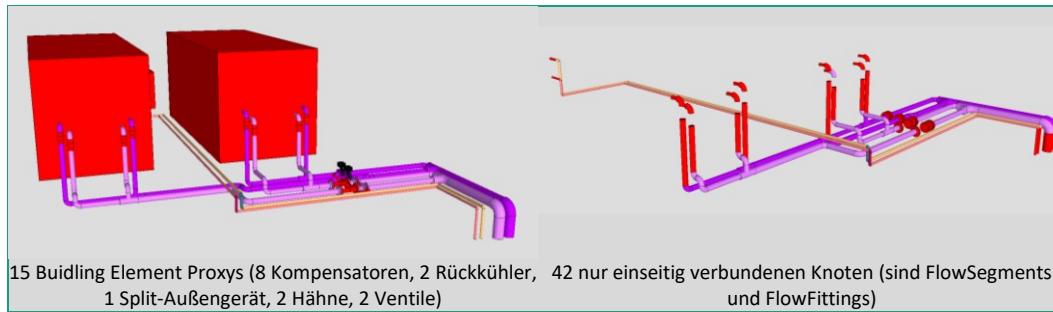


Abb. 54 Komponenten im DG-Modell, die im Originalmodell ohne Verbindung zur Nachbarkomponenten ausgespielt wurden

Alle notwendigen Aufbereitungsschritte sind in Tab. 47 zusammengefasst. Es ist zu unterscheiden zwischen zufälligen und systematischen *Fehlern* im Modell. Als *Fehler* wird in diesem Zusammenhang all das bezeichnet, was die Voraussetzungen gemäß Abschnitt 3.5.1 verletzt. Zufällige Fehler sind daran zu erkennen, dass sie nicht flächendeckend auftreten. Sie können durch gründliches Arbeiten ggf. nach einer Modellierungsanweisung behoben werden, sind aber bei manueller Modellerstellung nie ganz zu vermeiden. Es sind i.d.R. ausreichend Möglichkeiten vorhanden, derartige Fehler zügig zu erkennen und zu beheben.

Übersicht
Aufbereitung

Systematische Fehler sind bei derselben Arbeitsweise immer wieder zu erwarten, daher wurde der Aufwand für ihre Behebung eingeschätzt. Keiner der durchgeföhrten Aufbereitungsschritte hat einen so hohen Aufwand, dass er den Workflow unwirtschaftlich machen würde. Für die meisten systematischen Fehler wurden im Rahmen dieser Arbeit Skripte erarbeitet (Abschnitt 4.3.1).

Tab. 47 Aufbereitungsschritte für das Beispielfile

Kurzbeschreibung	Fehlertyp und Bewertung	Behebung: Notwendigkeit und Aufwand
überflüssige Elemente entfernen	Zufälliger Fehler: nur vereinzelt unnötige Elemente im Modell enthalten	Optional, Gering
alle Geschosse zusammenfügen	Systematischer Fehler	Optional, Einfach, durchgeführt mit simpleBIM
IFC-Version anpassen	Systematischer Fehler	verpflichtend, Gering, siehe 4.3.1.1
IFC-Klassen und predefinedTypes korrigieren für IfcBuildingElementProxies	Systematischer Fehler: beheben mit Modellierungsanweisung bzw. neuer Programmversion, IfcBuildingElementProxies werden vorrangig für Großkomponenten (Rückkühler, Kältemaschine, Umluftkühler, Lüftungsgeräte) erzeugt	verpflichtend, Mittel, siehe 4.3.1.2
Verbindungen an Geschossübergängen herstellen	Systematischer Fehler	verpflichtend, 4.3.1.3 Mittel, Zusammengehörigkeit kann anhand Geometrie leicht gefunden werden
Verbindungen und Ports an IfcBuildingElementProxies	Systematischer Fehler	verpflichtend, Hoch 692 von 6302 IfcElement
fehlerhafte Verbindungen entfernen	Zufälliger Fehler	verpflichtend, Mittel
Portbezeichnungen vergeben	Systematischer Fehler: ggf. durch bessere Exporteinstellungen behebbar	Optional, Gering, siehe 4.3.1.4
Zuordnung zu IfcBuildingStorey statt IfcSite	Systematischer Fehler: ggf. durch bessere Exporteinstellungen behebbar	Optional, Gering, siehe 4.3.1.5
Systemzuordnung korrigieren	Zufälliger Fehler: nur vereinzelt fehlende oder falsche Systemzuordnung	verpflichtend, Mittel, siehe 4.3.1.6
eindeutige, menschenlesbare Bezeichnungen vergeben	Systematischer Fehler: kann durch Modellierungsanweisung oder Exporteinstellungen ggf. behoben werden	Optional, Gering, siehe 4.3.1.7
Koordinaten runden	Systematischer Fehler: ggf. durch bessere Exporteinstellungen behebbar	Optional, Gering, siehe 4.3.1.8
Duplikate in der Serialisierung reduzieren	Systematischer Fehler aufgrund der Implementierung im Autorentool	Optional, Gering, siehe 4.3.1.8

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

Ergebnis der Aufbereitung

Das bereinigte Modell `25c.ifc`⁵⁰ enthält 4548 `IfcProducts`. Die Zuordnung zu den Klassen, Geschossen und Systemen zeigt Abb. 55.

Object Classes		IFC System	
Building	1	KK_Erzeugung	11
Building Storey	6	KK_Erzeugung_RL	220
▷ Chiller	1	KK_Erzeugung_VL	161
▷ Coil	4	KK_Rueckkuehlung	24
Distribution Port	0/9 275	KK_Rueckkuehlung_RL	227
▷ Filter	8	KK_Rueckkuehlung_VL	152
▷ Flow Instrument	37	KK1_BKA	1
▷ Heat Exchanger	2	KK1_BKA_RL	381
Model Information	1	KK1_BKA_VL	391
▷ Pipe Fitting	2 134	KK2_ULK	44
▷ Pipe Segment	1 955	KK2_ULK_RL	1 348
Project	1	KK2_ULK_VL	1 068
▷ Pump	9	KK3_Server	2
▷ Sensor	1	KK3_Server_RL	111
Site	1	KK3_Server_VL	128
▷ Tank	4	KK4_RLT	4
▷ Unitary Equipment	48	KK4_RLT_RL	129
▷ Valve	345	KK4_RLT_VL	148
Klassenzuordnung		Geschosszuordnung	
		Systemzuordnung	

Abb. 55 Zuordnungen aller `IfcProduct` des fertig aufbereiteten Modell (Screenshot aus simpleBIM)

Größenreduktion

Nach der inhaltlichen Nachpflege der Modelle gemäß Tab. 47 wurde anschließend die Dateigröße optimiert. Tab. 48 listet die wesentlichen Schritte und zeigt ihren Effekt auf die Datengröße und den Dateinhalt. Durch einfache Rundung der Koordinaten in der Datei auf Millimeter ergibt sich bereits eine signifikante Verkleinerung der Dateien um 11%. Dabei sind Millimeter immer noch deutlich unterhalb üblicher Bautoleranzen, auch in der grafischen Darstellung des Modells mit üblichen BIM-Viewern ist kein Unterschied erkennbar.

Anschließend wurden Duplikate von Ressourcenentitäten z.B. `IfcCartesianPoint`, `IfcDirection`, `IfcAxis2Placement` entfernt. Dadurch ändert sich an der semantischen Bedeutung des Dateiinhalts nichts, es handelt sich lediglich um eine technische Änderung in der Serialisierung. Dadurch lässt sich die Dateigröße nochmals erheblich auf 30 Prozent der Ursprungsgröße verringern.

⁵⁰ https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/25_Neustart_ab_23b_ohne_unwichtige_Systeme/25c.ifc

Tab. 48 Effekt der Größenreduzierung der IFC-STEP-Dateien am Beispiel EAS KLT (für alle Teilsysteme)

Schritt	Alle Systeme				Nur Serverkreis	
	Dateigröße IFC-STEP	Zeilen IFC-STEP	Anzahl Knoten im Graph	Dateigröße als ttl-File*	IFC-STEP	TTL-File
0 Original	239MB (100%) ⁵¹	4.427.506 (100%)	4.564 (100%)	Erzeugung mit der vorhandenen Hardware nicht möglich (OutOfMemory)		
1 Gerundete Koordinaten	214 MB (89%) ⁵²	(100%)	100%			
2 (Entfernung von Duplikaten von Ressourcenentitäten)	71MB (30%) ⁵³	1.359.898 (31%)	100%	793 MB (Erzeugung dauert 7min)	3MB ⁵⁴	42MB
3 Ergänzung Replacements für nicht-berechnungsrelevante Bauteile		110%	107%			
4 Entfernung nicht-berechnungsrelevante Bauteile		20%	23%			
5 Ergänzung 2D-Geometrie			23%			
6 Entfernung 3D-Geometrie	6MB (2%) ⁵⁵	72.268 (2%)	1.026 (23%)	46,7MB	0,2 MB ⁵⁶	0,7 MB

* Umwandlung mit IFC2OWL

5.1.3.2 Konverter 2 Schematizer

Anschließend wurden alle nicht-berechnungsrelevanten Komponenten, die zur Simulation lediglich einen Druckverlust beitragen würden, mit ihren benachbarten Elementen zu *replacements* zusammengefasst. Abb. 56 zeigt dies beispielhaft anhand des Anlagengraphen für den Serverkreis:

- A. Dargestellt sind dort zunächst alle Knoten im Originalgraph. Dabei handelt es sich um 241 Komponenten, die entsprechend ihren Klassen- und Typpzuordnung eingefärbt sind. Die Platzierung erfolgte anhand eines kraftbasierten-Layouts (siehe Abschnitt 3.5.4).

Die Teilabbildungen A und B zeigen denselben Graph in unterschiedlichen Layouts.

- B. Anschließend werden für Reihenschaltungen nicht-berechnungsrelevanter Bauteile die *replacements* ergänzt. Diese erhalten eine rote Färbung. Sie erhalten jeweils eine Bezeichnung bestehend aus *replacement_* und einer fortlaufenden Nummer. Dieser Schritt wurde für alle hydraulischen Sub-Systeme durchgeführt, weshalb die Nummern im Kühlkreis Server nicht fortlaufend sind.

- C. In einem letzten Schritt werden nun die Reihenschaltungen nicht-berechnungsrelevanter Komponenten entfernt (Teilabbildung D). Es verbleiben lediglich ca. 20% der Originalkomponenten, bestimmte Klassifizierungen/Typisierungen sind nun nicht mehr vorhanden.

In Teilabbildung E wurde dieser reduzierte Graph lediglich in einen anderen Layout dargestellt. Die Knoten wurden nun mit einem kraftbasierten Layout angeordnet, welches lediglich die verbliebenen Knoten berücksichtigt.

⁵¹ https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/20_System_korrigiert/all.ifc

⁵² https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/20_System_korrigiert/all_gerundeteKoordinaten.ifc

⁵³ https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/20_System_korrigiert/all_gerundeteKoordinaten_optimized.ifc

⁵⁴ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/Server_K.ifc

⁵⁵ https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/25_Neustart_ab_23b_ohne_unwichtige_Systeme/25y2.ifcs

⁵⁶ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/Server_F.ifc

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

Im Anhang ab S. 192 sind analoge Darstellungen für die anderen Teilsysteme enthalten.

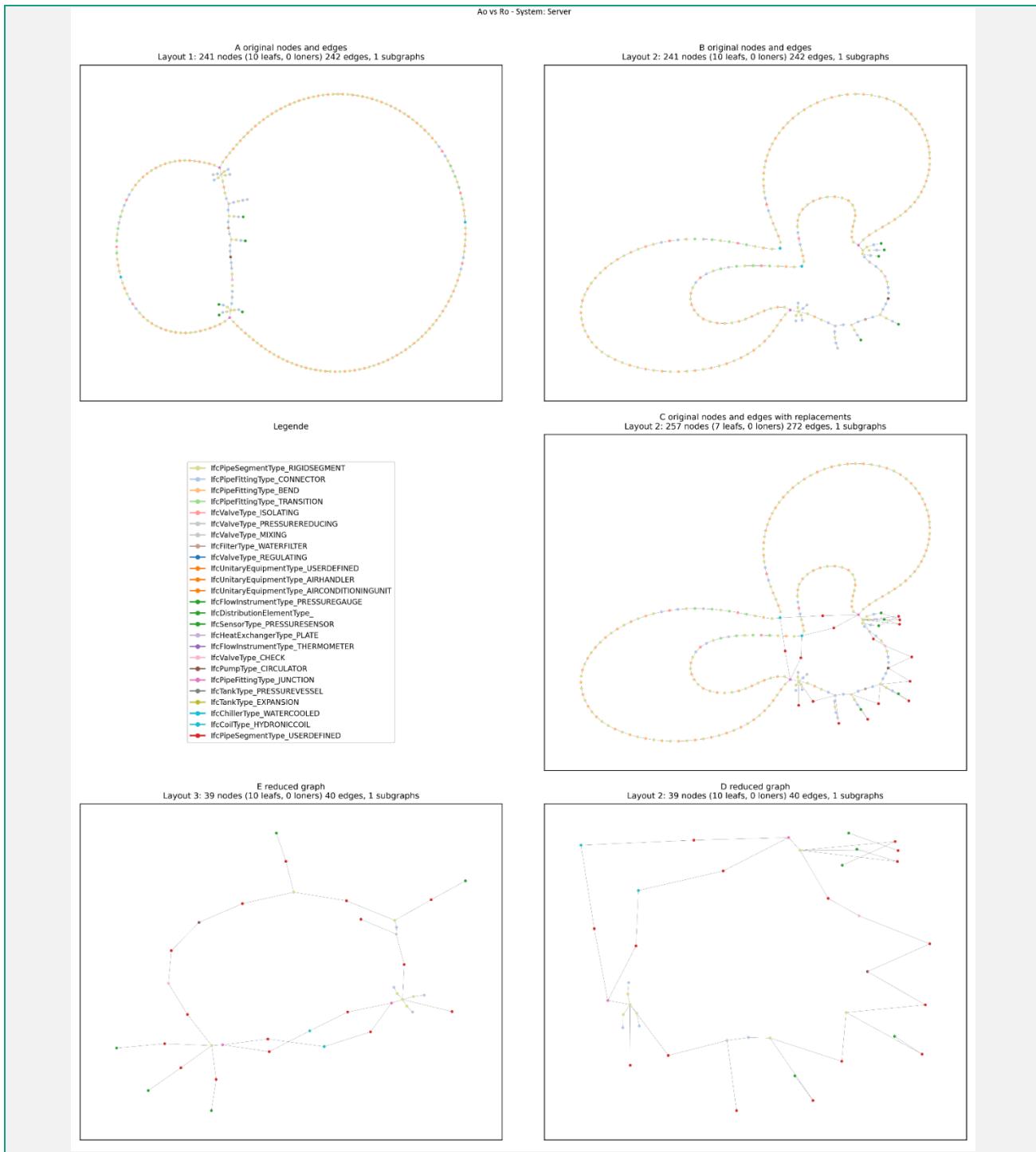


Abb. 56 Ableitung des Funktionsmodells aus dem Konstruktionsmodell am Beispiel des Serverkreises⁵⁷ (Darstellung mit matplotlib)

⁵⁷ IFC.IfcNetwork.IfcNetwork.figures_systemwise_compare_A_S in <https://github.com/ElisEck/MO-x-IFC/blob/main/PythonLib/IFC/IfcNetwork.py>

Abb. 57 zeigt beispielhaft einen Abschnitt aus mehreren Rohren und Bögen des Serverkreises, welcher durch ein Replacement ersetzt wurde.

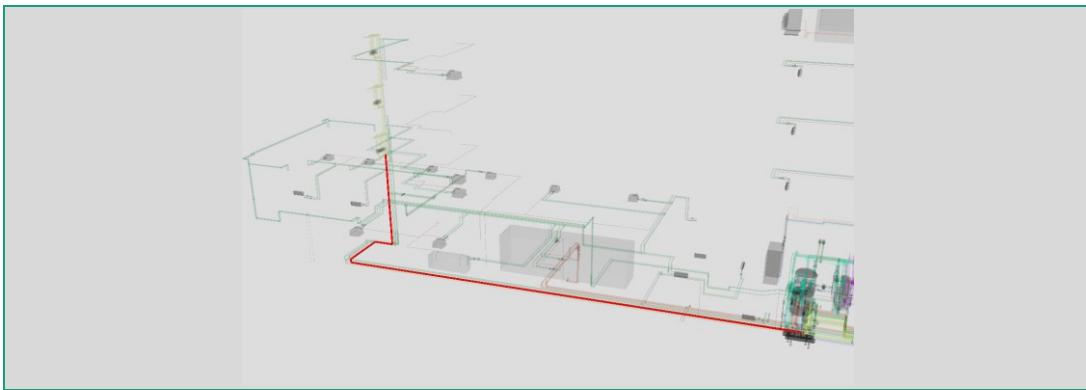
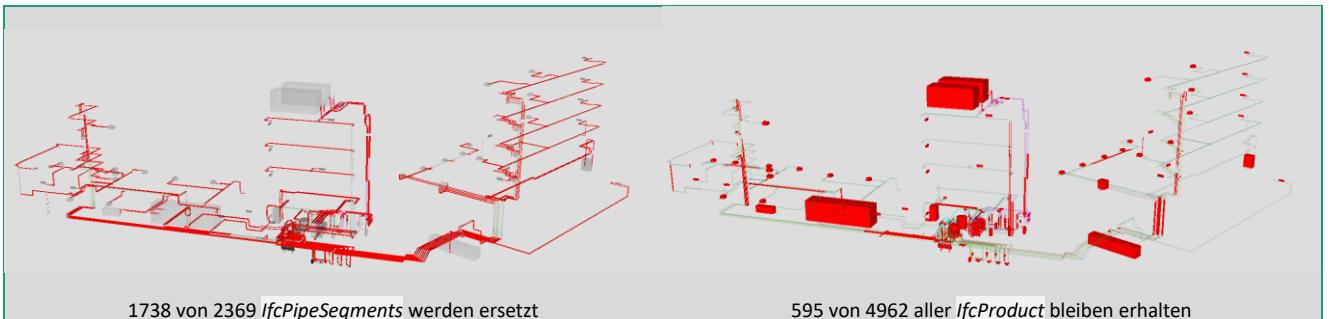


Abb. 57 *IfcGroup „replacement_12“ visualisiert in simpleBIM*

Für die Handhabung in IFC-Tools ist es i.d.R. notwendig, dass die Objekte eine Geometrie haben, um diese in einer 3D-Ansicht auswählen zu können. Daher wurde jedem *Replacement* die Geometrie eines der ersetzen Elemente zugewiesen. Die Gesamtgeometrie ist in Abb. 59 dargestellt und sieht im Vergleich zu Abb. 51 unvollständig aus. Die relevanten Verbindungen sind hierbei noch schlechter zu erkennen als vorher, jedoch kann die Dateigröße dadurch nochmals erheblich auf 20% reduziert werden (Schritt 4 in Tab. 48).

Zuweisung 3D-Geometrie

In Abb. 58 sind die ersetzen bzw. erhaltenen Objekte rot hervorgehoben.



1738 von 2369 *IfcPipeSegments* werden ersetzt

595 von 4962 aller *IfcProduct* bleiben erhalten

Abb. 58 Gegenüberstellung der 3D-Geometrie

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

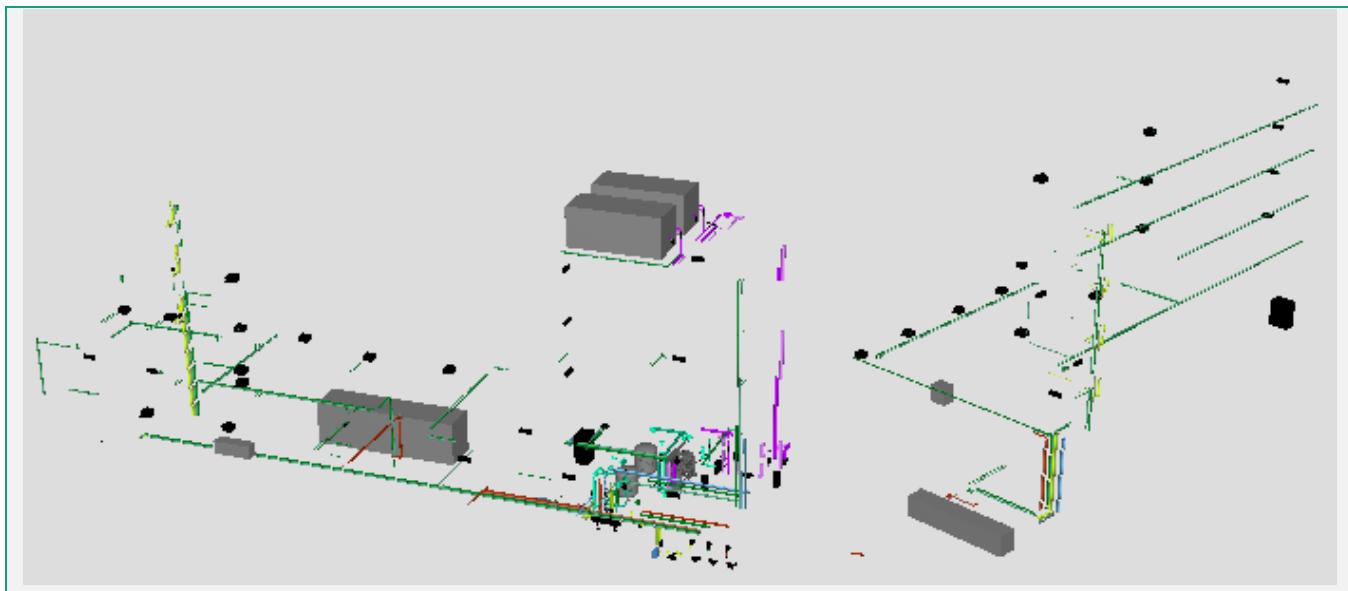


Abb. 59 Geometrie des Modells, welches nur noch die berechnungsrelevanten Bauteile enthält (entspricht XYZ_F.ifc in Abb. 33 und Tab. 29)

Abb. 60 zeigt die Anzahl der Knoten in den unterschiedlichen Vereinfachungsstufen für alle Teilsysteme des Kältemodells. Man erkennt, dass sich die Größe der Graphen durch den Ersetzungsalgorithmus wie erwartet erheblich verringert. Die dort angegebene Anzahl Knoten entspricht der Anzahl der Komponenten im späteren Modelica-Modell. In Abschnitt 3.5.2 wurde bereits herausgearbeitet, dass ca. 200 Komponenten im Modelica-Modell nicht überschritten werden sollten. Man erkennt, dass diese Anforderung durch den Ersatz der nicht-berechnungsrelevanten Komponenten erfüllt wird, eine Ausnahme bildet lediglich der größte Kühlkreis für die Umluftkühler. In Tab. 48 zeigt der Vergleich von Zeile 2 mit Zeile 6, dass sich die Dateigröße dadurch ebenfalls signifikant verkleinern lässt.

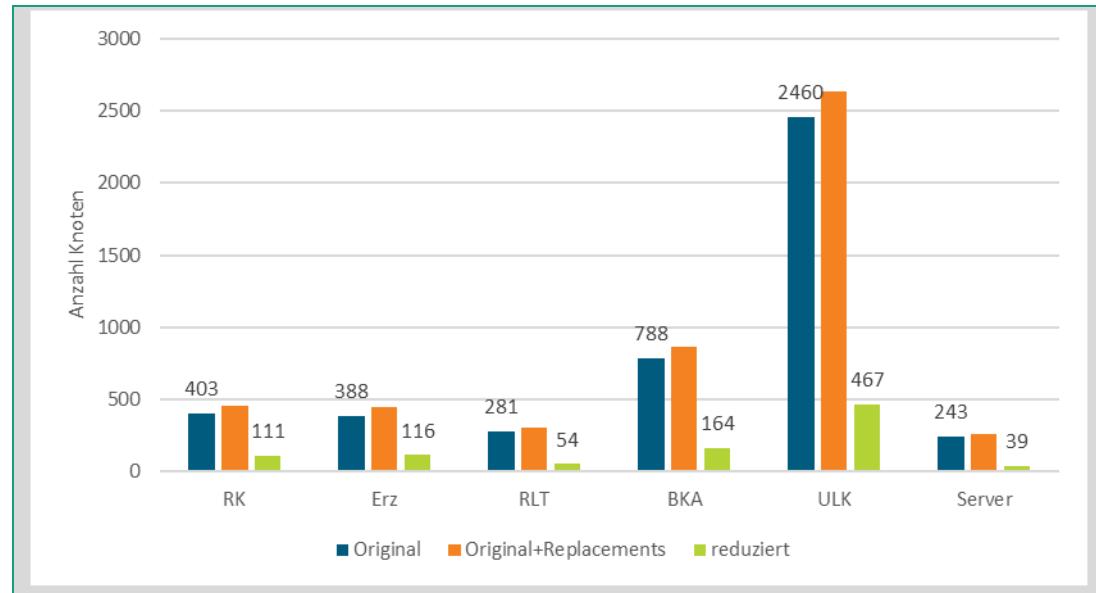


Abb. 60 Anzahl der Knoten in den verschiedenen Verarbeitungsstufen des Graphen

Tab. 49 Anzahl der Knoten in den verschiedenen Verarbeitungsstufen des Graphen

	Anzahl Knoten			Prozentuale Veränderung jeweils in Bezug auf das Original	
	Original	Original+ Replacements	reduziert	Ergänzung Repl.	Nur noch Replacements
Rückkühlung (RK)	403	459	111	114%	28%
Erzeugung (Erz)	388	443	116	114%	30%
Raumluftechnik (RLT)	281	301	54	107%	19%
Betonkernaktivierung (BKA)	788	865	164	110%	21%
Umluftkühler (ULK)	2460	2638	467	107%	19%
Server	243	259	39	107%	16%
Summe	4563	4965	951	109%	21%

Der Koordinatenursprung der 3D-Darstellung in der IFC-Datei befindet sich weit außerhalb des Gebäudes. Daher wurde die 2D-Repräsentation in der Nähe des Koordinatenursprung in der x-y (d.h. „Erdreichebene“) platziert, damit ist im vorliegenden Beispiel eine Überlappung zwischen beiden Repräsentation ausgeschlossen und es können beide parallel genutzt werden ohne sich zu verdecken. Der Maßstab der 2D-Darstellung wurde so gewählt, dass die 2D-Darstellung eine ähnliche Ausdehnung wie die 3D-Darstellung hat. Die Darstellung erfolgt stark vereinfacht mit Kreisen für die Knoten und Linien für die Verbindungen zwischen ihnen, siehe Abb. 61. Man erkennt das Layout aus Abb. 56-E wieder.

Darstellung des Ergebnisses im IFC

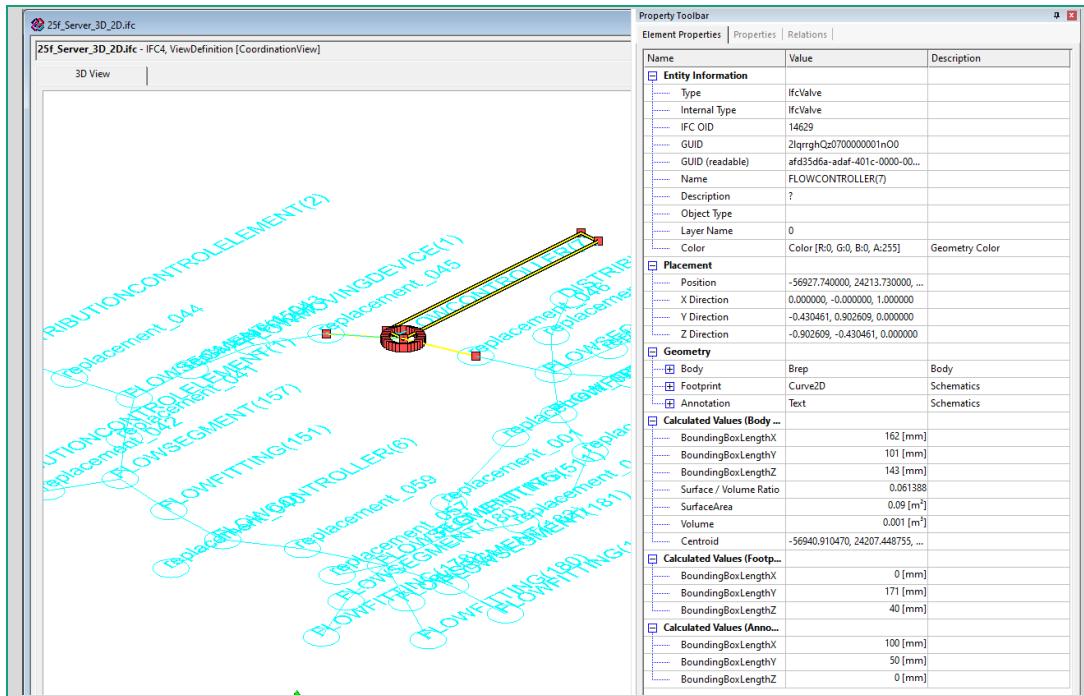


Abb. 61 schematische Darstellung des Serverkreis' im IFC⁵⁸ (entspricht XYZ_F.ifc in Abb. 33 und Tab. 29)
(Screenshot aus FZKViewer 6.5.1)

⁵⁸ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/debug/Server_F_3D_2D.ifc

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

5.1.3.3 Konverter 3 IFC2RDF

Die Überführung in einen KG erfolgt systemweise, da andernfalls nicht nur die Wissensgraphen, sondern auch das resultierende Simulationsmodell nicht mehr handhabbare Größen annehmen und dies ohnehin der in der Simulation erwarteten Modellteilung entspricht. Das IFC-Modell wird daher zunächst systemweise aufgeteilt. Falls nötig, kann die Dateigröße anschließend weiter verringert werden, indem geometrische Repräsentationen entfernt werden.

Die Überführung in einen KG erfolgt mittels des Tools IFC2RDF (Pauwels 2020), dafür wird die Serialisierung des KG als ttl gewählt. Der Vorgang ist sehr arbeitsspeicherintensiv, kann aber auch für die ca. 70MB große Gesamt-IFC-Datei problemlos und binnen weniger Minuten auf einem Standardrechner durchgeführt werden. Das entstehende ttl-File ist jedoch knapp 20-mal so groß wie die Original IFC-STEP-Datei. Lis. 33 zeigt die zugehörigen Aufrufe von Python-Skripten und der Java-Anwendung zur Transkription.

```
IntelliJ\MO-x-IFC>java -jar IFCtoRDF-0.4-shaded.jar "\exam-
    ples\BIM2SIM\A_Server\Server_F.ifc" "\exam-
    ples\BIM2SIM\A_Server\Server_ifc.ttl"
```

Lis. 33 Aufruf des Konverter 3 IFC2RDF⁵⁹

5.1.3.4 Konverter 4 Semantic Translator

Die weitere schriftliche Darstellung erfolgt aus Platzgründen nur für den Serverkreis.

Combine

Um aus dem in vorigen Schritt erstellten Wissensgraphen *XYZ_ifc.ttl* ein Simulationsmodell zu erzeugen, wird dieser zusammen mit dem IFC-Schema und dem IFC-AixLib-Alignment verarbeitet. Auch diese liegen in Form von Wissensgraphen vor. Die wichtigsten Kennzahlen dieses KG fasst Tab. 50 zusammen, man erkennt, dass das IFC-Schema mit Abstand der größte beteiligte KG ist, wenn das Beispiel-IFC-File geeignet aufbereitet⁶⁰ wurde (5.1.3.1 und 5.1.3.2).

Für das Reasoning ist es notwendig, alle KG zu einem KG zu kombinieren, diese Kombination wird im folgenden als *ABC.ttl* (A – Alignment, B – Beispiel, C – Schema) bezeichnet.

Tab. 50 Kennzahlen beteiligte Wissensgraphen (Beispiel Serverkreis)

	Dateiname allgemein	Im konkreten Beispiel ⁶¹	Axiome	Dateigröße bei Serialisierung als ttl (mittels Protege)
A Alignment	Aimaix.ttl	/TBox/alignments/AIMAix_0.5.0.ttl	190	36 kB
B Beispiel	XYZ_ifc.ttl	/examples/BIM2SIM/A_Server/Server_ifc.ttl	8.995	646 kB...800 MB*
C IFC-Schema	IFC4_0_2_1.ttl	/TBox/ontologies/2_IFC/IFC4_ADD2_TC1.ttl	20.641	2.619 kB
ABC		https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/semanticscripting/-/blob/master/2023-06-12Translator/TTL_ABC Alignment u Beispiel u Schema/mit Serverkreis/V17_A25_B25y4Server_C20.owl	29.782	-

* je nach vorheriger Aufbereitung des Modells und betrachtetem Teilsystem, siehe Tab. 48

Reasoning

OWLReady2 ermöglicht die Verwendung der Reasoner *Hermit* und *Pellet*. Beide sind nativ in Java implementiert und werden indirekt von *OWLReady2* in einer Java Virtual Machine ausgeführt. Tab. 51 enthält beispielhafte die Laufzeiten für das hier behandelte Beispiel der zwei Reasoner bei Aufruf innerhalb der *OWLReady2* und innerhalb des Ontologie-Editors *Protege*. Letzterer wurde in der Entwicklung des Prototypen immer wieder parallel verwendet, da die grafische Oberfläche

⁵⁹ Basispfad: <https://github.com/ElisEck/MO-x-IFC/>

⁶⁰ Das Einlesen des KG für das unaufbereitete Beispiel (ca. 800MB als ttl-Serialisierung) mittels *RDFlib* benötigt eine Laufzeit von ca. 30min und macht den Workflow dadurch praxisuntauglich langsam.

⁶¹ Basispfad, wenn nicht anders angegeben: <https://github.com/ElisEck/MO-x-IFC/blob/main/>

manche Dinge im Vergleich zu den Python-Bibliotheken vereinfacht darstellt. Man erkennt eindeutig, dass der Pellet Reasoner für den vorliegenden Anwendungsfall zu bevorzugen ist, da er deutlich kürzere Laufzeiten aufweist - unabhängig davon, innerhalb welcher Umgebung er verwendet wurde. Die Laufzeiten des *Hermit*-Reasoners lassen seine praktische Anwendung nicht zu.

Tab. 51 Laufzeiten Reasoning

Umgebung - Reasoner	Pellet	Hermit
Protege	<1min	erfolgloser Abbruch nach 10h
OWLReady2	~1min	erfolgloser Abbruch nach 10h

Lis. 34 zeigt einen Ausschnitt der geschlussfolgerten Triple. Darin ist z.B. die Zuordnung der *IfcPump_364* zur Modelica-Klasse *AixLib.Fluid.Movers.FlowControlled_m_flow* erkennbar, sowie die Zuordnung von zwei Ports zu der Pumpe.

```

<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPump_364> a aima:P_C,
    <https://w3id.org/aix/AixLib.Fluid.Movers.FlowControlled_m_flow>,
    moont:MComponent ;
    aimax:object_has_property <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_115>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_1167>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_1247>,
...
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_893>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_918> ;
    moon:hasPart <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcDistributionPort_366>,
        <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcDistributionPort_368> ;

```

Lis. 34 Ausschnitt aus geschlussfolgerten Triple KG XYZ_classes_mo.ttl in Turtle-Syntax⁶²

Wie bereits erwähnt, ist es nicht möglich, die Inhalte von *DataTypeProperties* durch Reasoning zuzuordnen, weiterhin unterstützen die Reasoner auch keine *PropertyChains* variabler Länge. Dafür kommen stattdessen *SPARQL INSERT* queries zu Einsatz, wie in Lis. 24 dargestellt. Für die Übertragung der Position wird das Python Skript aus Lis. 25 eingesetzt. Ein Ausschnitt aus dem sich ergebenden Wissensgraphen zeigt Lis. 35. Enthalten sind sind neben den geschlussfolgerten Triplen, welche bereits in Lis. 34 gezeigt wurden, auch „Modellparametrierungen“, welche zuvor in *DataTypeProperties* codiert waren z.B. die GlobalID und die Bezeichnung der Komponente.

Enhance_mo

⁶² https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/debug/Server_classes_mo.ttl

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

```
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPump_364> a aimaix:P_C,
  <https://w3id.org/aix/AixLib.Fluid.Movers.FlowControlled_m_flow>,
  moont:MComponent ;
  aimaix:object_has_property <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_115>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_1167>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_1247>,
...
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_893>,
<http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcPropertySingleValue_918> ;
  moont:comment "21qrrghQz0700000001nNz" ;
  moont:hasPart <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcDistributionPort_366>,
    <http://linkedbuildingdata.net/ifc/resources20230725_090232/IfcDistributionPort_368> ;
  moont:identifier "FLOWMOVINGDEVICE(1)" ;
  moont:placement_origin "{2572.1, 2310.9}"^^xsd:string Lis. 35 Ausschnitt aus Beispiel moont als KG XYZ_mo.ttl63
  (Turtle-Syntax), farblich hervorgehoben sind die Ergänzungen zu Lis. 34
```

5.1.3.5 Konverter 5 TTL2MO

In einem finalen Schritt erzeugt der Konverter *TTL2MO* aus dem ergänzten Wissensgraph gemäß Lis. 35 wieder ein natives Modelica-File. Abb. 62 zeigt den Diagram Layer dieses Modells.

⁶³ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/Server_mo.ttl

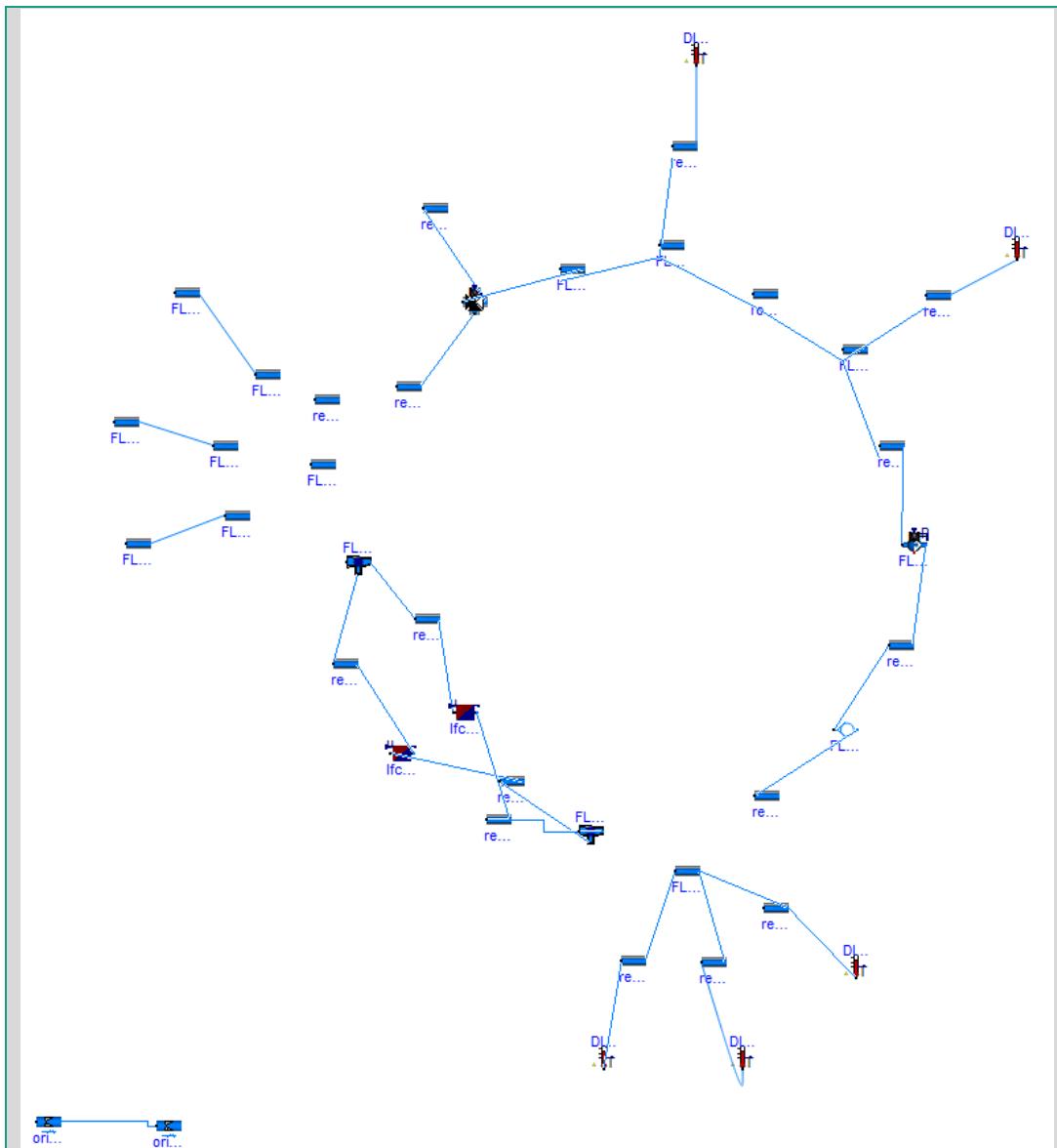


Abb. 62 Diagram Layer des erzeugten Modells (Screenshot aus Dymola)
(entspricht XYZ_raw.mo in Abb. 33 und Tab. 29)

In Abb. 62 sind zwei Komponenten zu erkennen, welche ohne Verbindung zu anderen Komponenten dargestellt sind. Dabei handelt es sich nicht um Darstellungsfehler, sondern es sind tatsächliche keine Verbindungen vorhanden. Vergleicht man Abb. 62 mit dem nachbearbeiteten Modell in Abb. 66 erkennt man, dass diejenigen Komponenten ohne jegliche Verbindung sind, welche tatsächlich mehr als 3 Verbindungen aufweisen. Abb. 63 zeigt ein solches Beispiel im Konstruktionsmodell. Man erkennt ein Rohr, welches fünf Verbindungen aufweist: an jedem Ende eine, sowie drei weitere für die Anbringung von Messinstrumenten. Für eine korrekte Modellierung in Modelica müsste dieses Rohr aufgeteilt werden in drei T-Stücke, damit würden dann die Druck- und Wärmeverluste zwischen den Anschlusspunkten der drei Messinstrumente modelliert. Jedoch ist deren Modellierung mit den zur Verfügung stehenden Bibliotheks-Komponenten für eine derartig kleinräumliche Auflösung nicht genau genug, aber auch nicht notwendig. Daher wurde von einer Implementierung abgesehen, stattdessen wurden in der manuellen Nachbearbeitung, wie in Abb. 66 dargestellt, die Anschlüsse aller 3 Messinstrumente

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

auf einen Modelica-Port verbunden, dies ist ausreichend genau. Der Konstrukteur hat diese Vereinfachung implizit vorweggenommen, indem er dort keine T-Stücke modellierte.

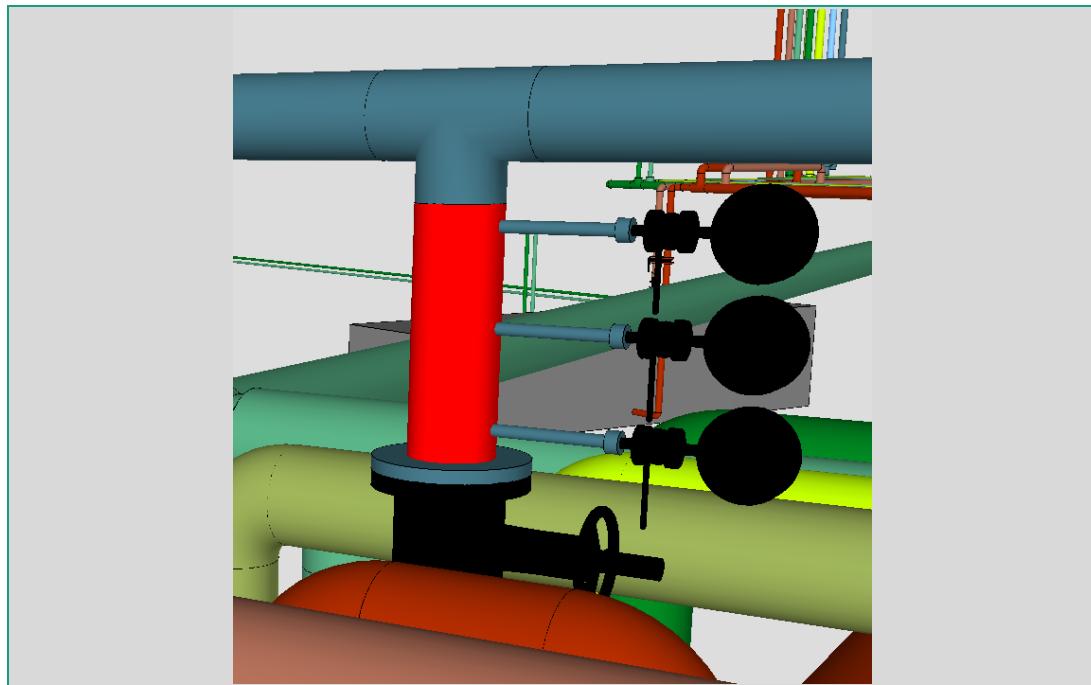


Abb. 63 Ausschnitt aus Abb. 51, rot markiert replacement_045 (Flowsegment(163))

5.1.3.6 Schritt 6: Überführung des Strukturmodells in ein rechenfähiges Simulationsmodell

Das in Abb. 62 dargestellte Modell ist noch nicht rechenfähig sondern bedarf einiger Nacharbeiten. Tab. 52 fasst diese zusammen, alle aufgeführten Themen müssen behoben werden, um die Rechenfähigkeit der Modells herzustellen. Es wird dabei unterschieden zwischen Nachbearbeitungen, welche potenziell noch durch eine bessere Implementierung eliminiert werden könnten und solchen, die bewusst dem Simulationsingenieur überlassen werden sollten, da dafür ingenieurtechnische Abwägungen nötig sind.

Tab. 52 Nachbearbeitung der MO-Datei

Kurzbeschreibung	Bewertung	Aufwand Behebung (allgemeine Einschätzung und am Beispiel Serverkreis)
Verbindungen für Komponenten mit mehr als 3 Ports ergänzen	Weist auf potentiellen Fehler im IFC-Modell hin – Simulationsingenieur	Mittel, 10 Stück
Verbindungen mit falscher Portbezeichnung ersetzen	Weist auf Fehler im IFC-Modell hin - Simulationsingenieur	Gering, 0 Stück
Druckverlustparametrierung für Ventile, Messinstrumente, T-Stücke anpassen	Implementierung möglich	Mittel, 9 Stück
Parametrierung HeaterCooler mit Nennleistung	Implementierung möglich	Gering, 1 Stück
Redeclare package Medium statt Medium =	Implementierung möglich	Gering, im Texteditor, suchen/ersetzen
Ergänzung Druckboundary (Druckhaltung), Hydraulische Anbindung an andere Teilsysteme	Notwendigkeit entsteht aufgrund der Aufteilung des Kaltwassersystems in mehrere Simulationsmodelle – Randbedingungen sollten vom Simulationsingenieur festgelegt werden	Gering, 2 Stück
RealInputs (Regelsignale) ergänzen für Pumpe, 3WV, HeaterCooler	Notwendigkeit entsteht aufgrund der Nichtabbildung der Regelungstechnik im IFC-Modell, Modellansätze sollten vom Simulationsingenieur festgelegt werden	Gering, 4 Stück

Dazu gehören Anpassungen, die wegen des Herausschneidens des Teilsystems als Modelica-Modell erforderlich werden: Es fehlt die Druckhaltung und auch die Einspeisung aus dem Erzeugerkreis. Mit welchen Randbedingungen diese abstrahiert werden sollen, ist eine ingenieurtechnische Abwägung, welche dem Simulationsingenieur überlassen wird. Zwar könnte die Stelle der Anbindung an das restliche System mit einem Aufwand automatisch ermittelt werden, jedoch nicht die inhaltlichen Randbedingungen. Daher wird von einer Implementierung abgesehen.

Die automatische Modellerzeugung beschränkt sich auf hydraulische Komponenten, dementsprechend sind die Signaleingänge der steuerbaren Komponenten (Pumpe, 3-Wege-Ventil und Verbraucher) unbelegt. Gebäudeautomationskomponenten oder gar ihre interne Logik sind in den verwendeten IFC-Modellen nicht enthalten, können dementsprechend auch nicht zu Modelica übertragen werden. Es obliegt dem Simulationsingenieur die Eingänge sachgerecht mit Signalen zu belegen, dafür stehen z.B. zahlreiche Regelungskomponenten in den IBPSA-Libraries zur Verfügung.

Da Modelica mit den gewählten Komponenten eine thermohydraulische Berechnung durchführt, ist für alle Komponenten eine Parametrierung hinsichtlich des Strömungswiderstands notwendig, dafür ist i.d.R. der Druckverlust bei einem bestimmten Massestrom anzugeben. Es werden dafür im Modell globale Parameter für Nenndruckverlust dp_{nom} und Nennmassestrom mp_{nom} angelegt. Alle Komponenten werden zunächst mit diesen initialisiert. Dies stellt sicher, dass alle Komponenten zueinander in sinnvoller Größenordnung stehen. Der Simulationsingenieur kann anschließend diese Werte einmalig setzen und für die wesentlichen Komponenten die Druckverluste manuell überschreiben. Für alle nicht manipulierten Komponenten verbleibt damit als Vorgabewert ein geringer Druckverlust in der richtigen Größenordnung (siehe Abb. 64 und Abb. 65). Für die thermohydraulische Berechnung sind auch Kenntnisse über die Stoffwerte des strömenden Mediums notwendig. Es kann i.d.R. davon ausgegangen werden, dass dieses für alle Komponenten gleich ist. Daher wird auch hierfür ein globaler Parameter (in Form eines *replaceable package*) definiert, welcher allen Komponenten zugewiesen wird. Die Standardbelegung für das *Medium* ist *AixLib.Media.Water*, welches flüssiges Wasser stark vereinfacht mit konstanten Stoffwerten abbildet (genutzt werden die Werte für 20°C), alternative Implementierungen z.B. *AixLib.Media.Specialized.Water.TemperatureDependentDensity* stehen in den einschlägigen Bibliotheken ebenfalls zur Verfügung. Abb. 64 und Abb. 65 zeigen die beschriebene mehrstufige Parametrierung.

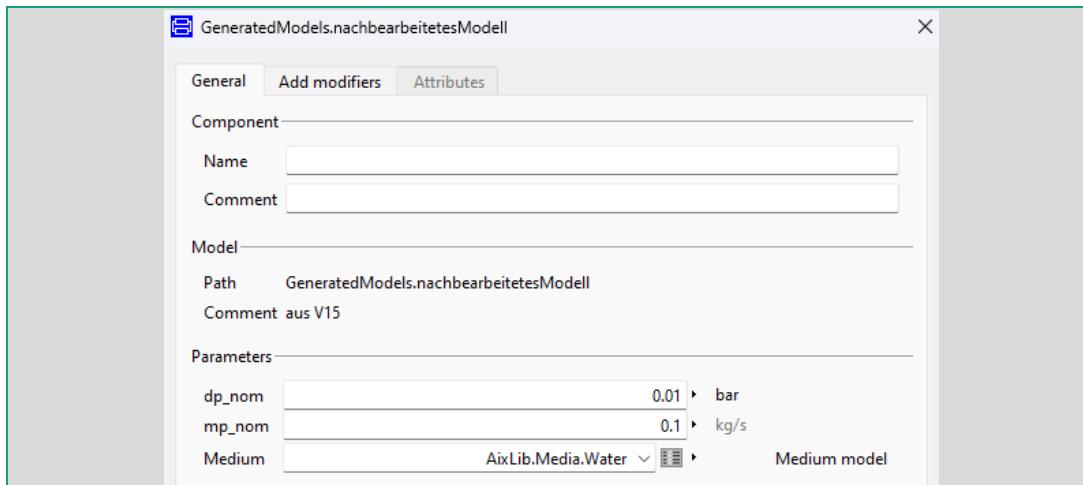


Abb. 64 globale Parameter des erzeugten Modells (Screenshot aus Dymola)

5 Experimente

5.1 UseCase 1: BIM2SIM

5.1.3 Erprobung

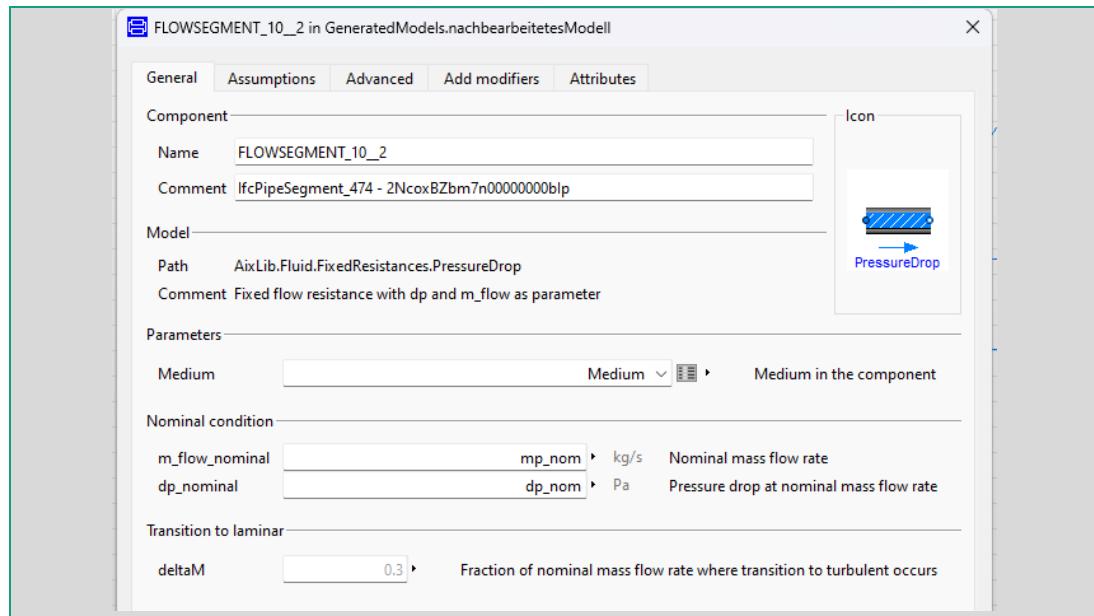


Abb. 65 Parameter einer Modellkomponente (Screenshot aus Dymola)

Abb. 66 zeigt das finale nachbearbeitete Modell. Man erkennt die manuell nachgepflegten Verbindungen am rechtwinkligen Verlauf. Dieses automatische Routing realisieren die Simulationsumgebungen. Im Beispiel wurde Dymola verwendet, OpenModelica gewährleistet dies ebenso.

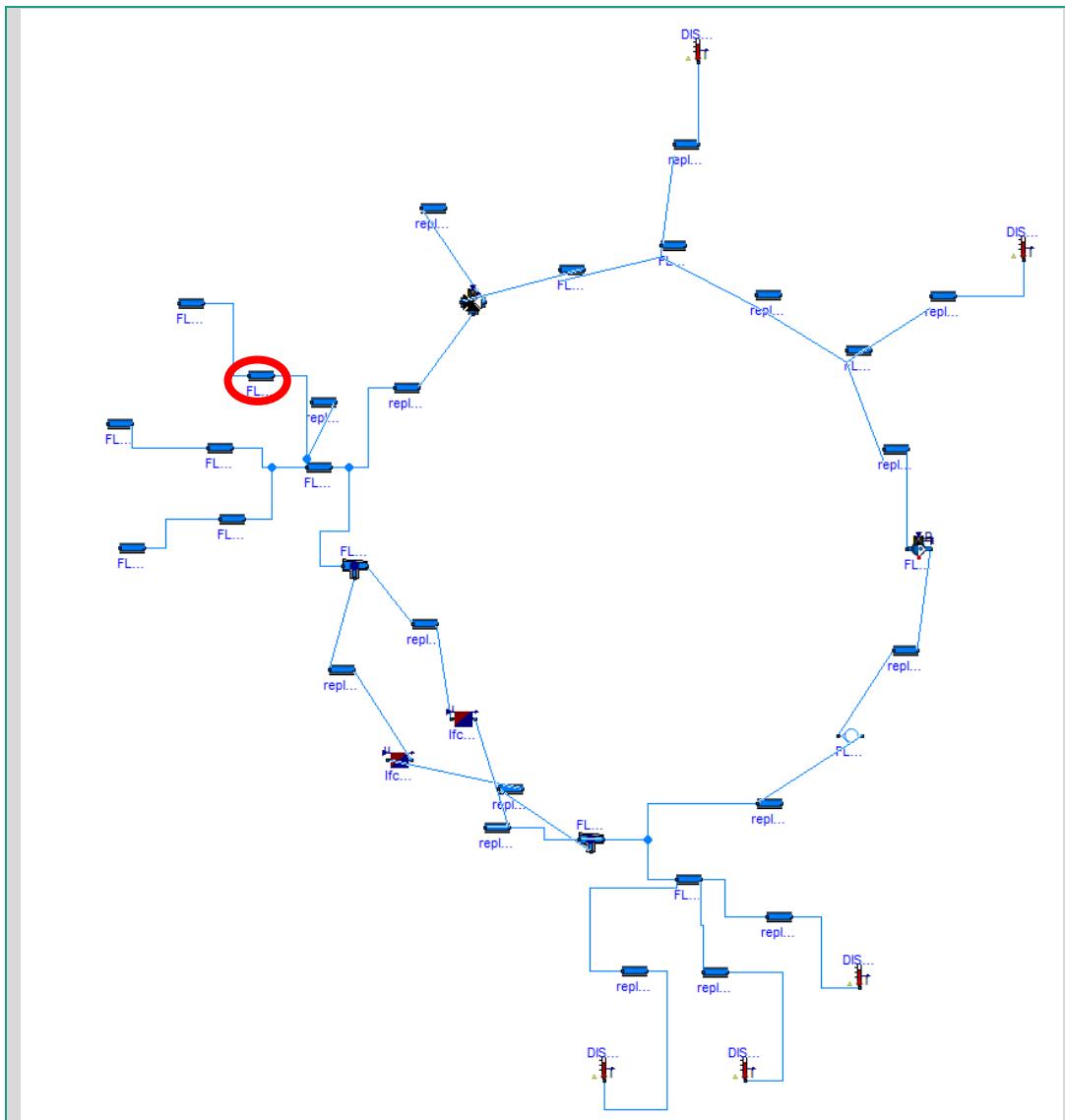


Abb. 66 Diagram Layer des nachbearbeiteten Modells (entspricht XYZ.mo in Abb. 33 und Tab. 29) hervorgehoben FLOWSEGMENT_10_2 (dessen Parameterdialog in Abb. 65 gezeigt ist)
(Screenshot aus Dymola)

Der Gesamtaufwand für die Nachbearbeitung des Modells in Abb. 62 zum Zustand in Abb. 66 lag bei ca. 2 h incl. Fehlersuche und wird daher für Wiederholungsfälle (bei anderen Modellen) auf wenige Minuten geschätzt. Die Dymola-Tools unterstützen dabei mit aussagekräftigen Fehlermeldungen.

5.1.4 Auswertung

Die Überführung des Konstruktionsmodells, welches als IFC-STEP-Datei vorlag, in ein Simulationsmodell funktioniert mit Hilfe der implementierten bzw. bereits vorhandenen Konverter. Es wurde kein vollautomatischer Workflow realisiert, da für die in Abb. 48 und Tab. 46 dargestellten Zwischenstände weitere Verwendungen denkbar sind (siehe Abschnitt 6.4.1).

Der mit dem UseCase verbundene Aufwand entsteht in der Vorbereitung der IFC-Konstruktionsmodelle und in der Nachbereitung des Strukturmodells zu einem rechenfähigen Simulationsmodell.

Wie bereits erwähnt, wurde mit IFC-Dateien gearbeitet, welche NICHT entsprechend standardisierten BIM-Workflows konform zu Auftraggeber-Informations-Anforderungen erstellt wurden. Der Vorbereitungsaufwand ließe sich durch das Einhalten von Modellierungsanforderungen deutlich verringern, jedoch verursachte keiner der durchgeführten Aufbereitungsschritte hat einen so hohen Aufwand, dass er den Workflow unwirtschaftlich machen würde. Für die meisten systematischen Fehler wurden im Rahmen dieser Arbeit Skripte erarbeitet (Abschnitt 4.3.1), um den Aufwand in Zukunft zu verringern.

Es war nicht Ziel der Experiments, automatisch ein rechenfähiges Simulationsmodell zu erstellen, vielmehr sollte eine valide Modellstruktur mit geeigneter Vorparametrierung entstehen, beides konnte erfolgreich nachgewiesen werden. Der notwendige Nachbereitungsaufwand wurde dokumentiert. Bei Nutzung einer geeigneten Simulationsumgebung, welche die Nacharbeit mit aussagekräftigen Fehlermeldungen unterstützt, ist dies in wenigen Minuten zu bewerkstelligen. Der Aufwand für die Bereitstellung einer validen Modellstruktur wurde durch die Automatisierung ebenfalls auf wenige Minuten reduziert.

Der Workflow addressiert nicht die eigentliche Arbeit des Simulationsingenieurs: das inhaltliche Anpassen des Simulationsmodells, um einen möglichst guten Entwurf zu erhalten. Vielmehr werden die dazu notwendigen Vorarbeiten automatisiert, so dass für diese Kernaufgabe mehr Zeit verbleibt.

5.2 UseCase 2: SIM2BIM

5.2.1 Überblick

Der UseCase SIM2BIM dient der Nachnutzung einmal erstellter Simulationsmodelle im Plausionsprozess. Abb. 67 zeigt das Zusammenwirken der Konverter im UseCase SIM2BIM. Die Unterscheidung zwischen den beiden nativen Modelica-Dateien *XYZ_raw.mo* und *XYZ.mo* ist für diesen UseCase weniger relevant: Der *MoTTL-Transcriptor* kann sowohl auf ein rechenfähiges Simulationsmodell, als auch auf ein nicht rechenfähiges Strukturmodell angewendet werden und erzeugt einen Wissensgraphen mit dem *MoOnt*-Vokabular.

Die Überführung zwischen den beiden Domänen erfolgt mittels *Procedural Translation*. Dies dient dem experimentellen Vergleich mit der alternativen Implementierung als *Semantic Translation*, wie sie im Abschnitt 5.1 implementiert wurde.

Anschließend erfolgt die Transkription in eine IFC-STEP-Datei mit dem Konverter *RDF2IFC*. In einem finalen Schritt, wird diese Datei durch den *Voluminizer* mit Platzhaltergeometrien angereichert.

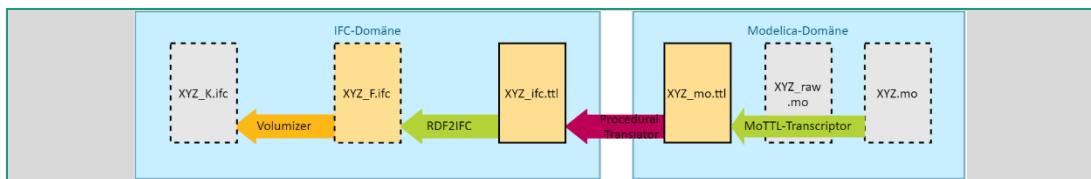


Abb. 67 Zusammenspiel der Komponenten für den UseCase SIM2BIM

Tab. 53 Bezeichnung der Zustände im Experiment zu UC1 BIM2SIM

Allgemeine Bezeichnung	Pfad im digitalen Anhang
	Basispfad: https://github.com/ElixEck/MO-x-IFC/tree/main/examples/SIM2BIM/D_HIL/
XYZ.mo	/HIL/HIL_flat.mo
XYZ_mo.ttl	/hil_mo.ttl
XYZ_ifc.ttl	/hil_ifc.ttl
XYZ_F.ifc	/hil_F.ifc
XYZ_K.ifc	/hil_F_3D_2D.ifc

5.2.2 Demonstrator Versuchsstand EAS

Die experimentelle Erprobung erfolgt am Beispiel des Simulationsmodells eines Versuchsstands, welches in Abb. 68 dargestellt ist. Aufgrund der komplexen Hydraulik ist für derartige Systeme ein simulationsbasierter Entwurf sinnvoll.

Das Beispielmodell enthält hauptsächlich hydraulische Komponenten wie Rohrstücke, T-Stücke, Pumpen und Speicher. Weiterhin sind zwei Wärmeübertrager (Beschriftung mit „1“ in der Abbildung) enthalten, die die Anlage in drei hydraulische Kreise unterteilen. Für jeden hydraulischen Kreis ist eine Druckhaltung (2) vorhanden. Die Anbindung an das übergeordnete Gebäudesystem wird durch eine Temperatur- und Druckrandbedingung abgebildet (3). Die Prüflinge werden durch ideale Verbraucher (4) modelliert. Die Steuerungs- und Regelungslogik ist im Modell nicht enthalten, stattdessen sind alle Steuereingänge mit fixen Werten (5) belegt. Hydraulische Verbindungen sind – wie es in der MSL vorgesehen ist – im Diagram-Layer als hellblaue Verbindungen (6) enthalten, Regelsignale analog dazu als dunkelblaue Verbindungen (7).

Im Beispielmodell werden einige benutzerdefinierte Komponenten verwendet, welche aus einer internen Library namens *LibEAS* stammen. Diese Komponenten enthalten jedoch i.d.R. nur geringfügige Ergänzungen im Icon-Layer im Vergleich zu ihren Ursprungsmodellen, die aus der *AixLib* stammen. Die benutzerdefinierten Bibliotheks-Komponenten erben (*extends*) von diesen Ursprungsmodellen aus der *AixLib*, daher können die Zuordnungen welche in Bezug auf die *AixLib* formuliert wurden, für das Beispiel ohne weiteres verwendet werden.

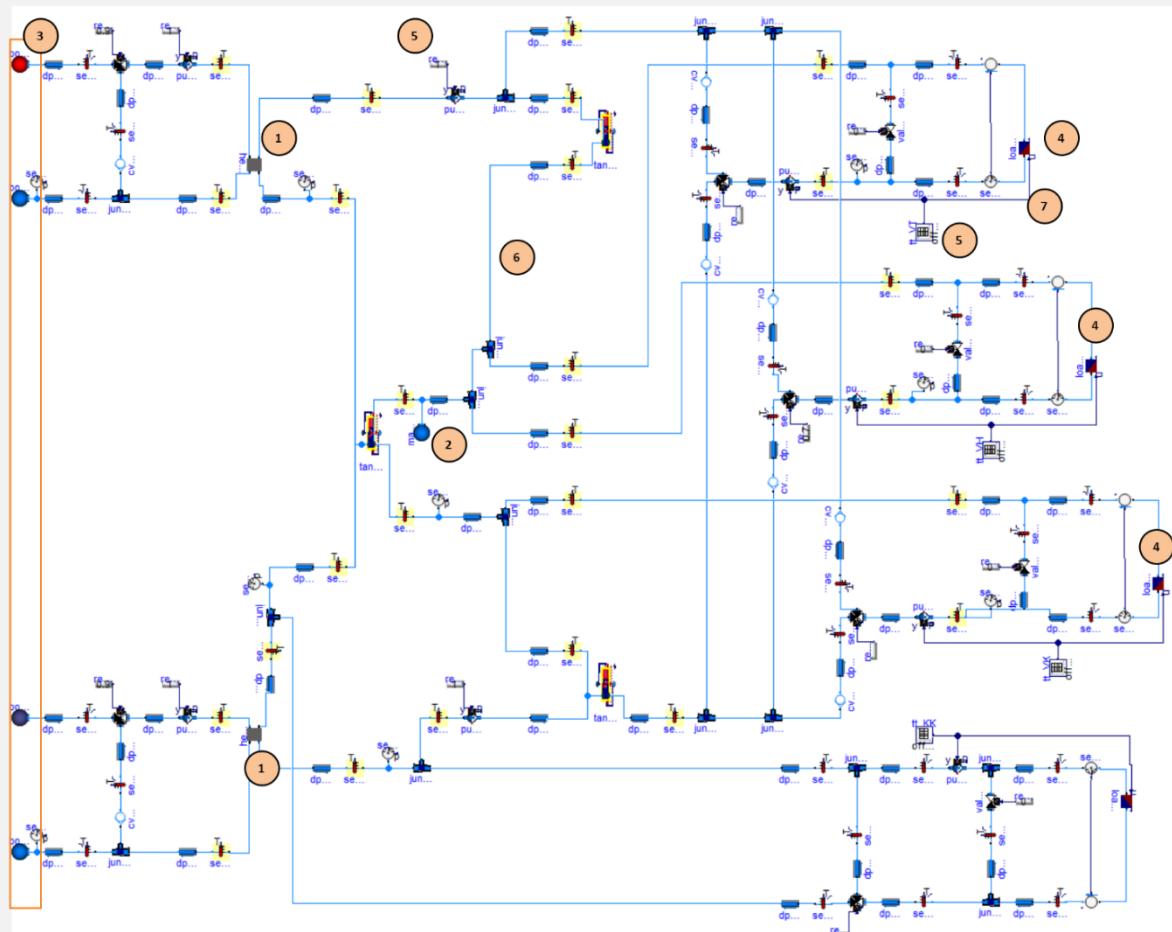


Abb. 68 Demonstrator Versuchsstand EAS (entspricht XYZ.mo in Abb. 33 und Tab. 29)

5.2.3 Erprobung

5.2.3.1 MoTTL-Transcriptor

Das Modell wurde einzeln in einer Modelica-Library gespeichert. Da der MoTTL-Transcriptor immer vollständige Libraries verarbeitet, vermeidet dies unnötige Triple im Wissensgraphen für die Übersetzung. Lis. 36 zeigt einen Ausschnitt des vom *MoTTL-Transcriptor* erzeugten Wissensgraphen mit MoOnt-Vokabular (entspricht dem „Mo-KG“ aus Tab. 29). Dieser enthält alle für die Übersetzung wesentlichen Information:

Man erkennt zunächst die notwendigen Imports der zu den Modelica-Libraries gehörigen Wissensgraphen (orange, siehe Abschnitt 3.7). Es werden die Vererbungsbeziehungen *moont:extends* und die Modellverschachtelung *moont:containedIn* in den Graph übertragen. Die Relation *moont:hasPart* verknüpft die Modellkomponenten mit dem Modell. Außerdem sind die Bezeichnungen *moont:identifier*, die Modifikationen *moont:modification* und die Platzierung des Symbols im Diagram-Layer *moont:placement_origin* enthalten.

Für die Übersetzung in ein IFC-Modell unwesentliche Informationen wie z.B. der Verlauf der Verbindungsleitungen, die Optik der Icons und die Dokumentation sind i.d.R. als Annotation im Modell enthalten (siehe Abschnitt 2.2.4) und werden vom *MoTTL-Transcriptor* bewusst nicht in den Wissensgraphen übertragen.

```

@prefix moont: <http://w3id.org/moont#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/2000/01/rdf#> .
@prefix hil: <http://www.eas.iis.fraunhofer.de/hil#> .
@prefix aix: <http://www.eas.iis.fraunhofer.de/aix#> .
@prefix libeas: <http://www.eas.iis.fraunhofer.de/libeas#> .

hil:    rdf:type owl:Ontology ;
        owl:imports          moont:,
                    aix:,
                    libeas:.

hil:HIL.HIL_flat.Medium moont:containedIn hil:HIL.HIL_flat ;
    rdfs:subClassOf moont:MPackage .

hil:HIL moont:stringComment "temporäres Package zur Freistellung von HIL flat"^^xsd:string ;
    rdfs:subClassOf moont:MPackage .

hil:HIL.HIL_flat moont:containedIn hil:HIL ;
    moont:extends msl:Modelica.Icons.Example ;
    moont:hasPart hil:HIL.HIL_flat.bou_EH_1,
        hil:HIL.HIL_flat.dp_EH_1,
        ...
        hil:HIL.HIL_flat.senT_VS_1,
        ...
        hil:HIL.HIL_flat.val_VT_5 ;
    moont:stringComment "kompletter HIL Versuchsstand (ohne Regelung) - ohne Submodelle"^^xsd:string ;
    rdfs:subClassOf moont:MModel .

hil:HIL.HIL_flat.dp_EH_1 a moont:MComponent,
    libeas:LibEAS.Sensors.PressureDrop_Display ;
    moont:hasPart hil:HIL.HIL_flat.dp_EH_1.dp_nominal,
        hil:HIL.HIL_flat.dp_EH_1.m_flow_nominal,
        hil:HIL.HIL_flat.dp_EH_1.port_a,
        hil:HIL.HIL_flat.dp_EH_1.port_b ;
    moont:identifier "dp_EH_1" ;
    moont:modification "redeclare package Medium_gleich_Medium"^^xsd:string ;
    moont:placement_origin "{-530,570}"^^xsd:string .

```

Lis. 36 Ausschnitt aus *HIL_mo.ttl*⁶⁴ (Imports farbig hervorgehoben) (entspricht *XYZ_mo.ttl* in Abb. 33 und Tab. 29)

⁶⁴ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2BIM/D_HIL/hil_mo.ttl

5.2.3.2 Procedural Translator

Dieser Mo-KG wurde anschließend mit dem *Procedural Translator* verarbeitet. Dabei ist es wichtig, dass die in Lis. 36 orange hervorgehobenen Importe von der Engine „aufgelöst“ werden können, d.h. dass der Import der in den Library-Ontologien enthaltenen Triple erfolgen kann. Dies wird gewährleistet, indem der Engine der Pfad zu den Importen bekannt gemacht wird, alternativ wird versucht die angegebenen URLs aufzulösen. Ohne den Import wäre die Übersetzung für die aus der LibEAS stammenden Komponenten unmöglich, da sie nicht direkt aus der AixLib stammen, für die aber das Mapping (siehe Lis. 30) formuliert ist.

Alle Komponenten, für die kein Mapping festgelegt ist, bleiben bei der Übersetzung außen vor, dazu gehören insb. die Komponenten, welche die konstanten Steuersignale generieren *Modelica.Blocks.Sources.RealExpression* und *Modelica.Blocks.Sources.TimeTable*. In Anbetracht der beschränkten Eignung und dementsprechend geringen Verbreitung des IFC-Formats für die Abbildung von Regelungslogik ist dies für den untersuchten Beispielfall keine signifikante Einschränkung. Weitergehende Überlegungen dazu finden sich in Abschnitt 6.4.1.2.

In Lis. 37 wird anhand einer ausgewählten Modell-Komponente *senT_VS_1* (Temperatursensor) die Grundstruktur der erzeugten Tripel aufgezeigt. Der entstandene Wissensgraph *HIL_IFC.ttl* enthält Wissen über die IFC-Klasse einer Komponente, die ihr zugehörigen Ports, deren Verbindungen und außerdem wichtige Eigenschaften, die in Modelica eingepflegt wurden. Dazu gehört beispielsweise das Anlagenkennzeichen, welches in Modelica als *stringComment* eingepflegt wurde.

In Abschnitt 3.4 wurde als wesentliche Anforderung an alle Konverter formuliert, dass die eindeutigen Identifikatoren aller Komponenten immer erhalten bleiben. Dies wurde zusätzlich auch für den *stringComment* umgesetzt, da dieser dem Nutzer eine Möglichkeit bietet, externe Identifikatoren (z.B. aus einem Anlagenkennzeichnungssystem) in ein Modelica-Modell einzufügen, obwohl er gemäß dem Sprachstandard MLS (siehe 2.2.2.4) keine Identifikator-Funktion hat.

5 Experimente

5.2 UseCase 2: SIM2BIM

5.2.3 Erprobung

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix express: <https://w3id.org/express#> .
@prefix hil: <http://www.eas.iis.fraunhofer.de/hil#> .
@prefix ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC4/ADD2_TC1/OWL#> .
@prefix list: <https://w3id.org/list#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

Header
hil: dcterms:creator "C:/_DATEN/WORKSPACES/PyCharm/SemanticScripting/2023-12-20 Modelica to IFC procedural with ports/04_generateIFC_classes_ports.py executed by eckstaedt on troll 918" ;
dcterms:issued "2024-01-05 12:56" ;
dcterms:source "MO-x-IFC/src/test/java/output/hil_moont.ttl" ;
dcterms:title "HIL" ;
owl:imports ifc: .

Klassenzuordnung und ausgewählte Attribute
hil:HIL.HIL_flat.sent_VS_1 a ifc:IfcSensor ;
ifc:globalId_IfcRoot hil:IfcGloballyUniqueId_291 ;
ifc:name_IfcRoot hil:IfcLabel_292 ;
ifc:predefinedType_IfcSensor ifc:TEMPERATURESENSOR .
hil:IfcGloballyUniqueId_291 express:hasString "2A5Fx9oT9yBXMs__XbTRX" .
hil:HIL.HIL_flat.sent_VS_1.port_b a ifc:IfcDistributionPort .

Portzuordnung
hil:IfcRelNests_2435 a ifc:IfcRelNests ;
ifc:globalId_IfcRoot hil:IfcGloballyUniqueId_2437 ;
ifc:relatedObjects_IfcRelNests hil:IfcObjectDefinition_List_2436 ;
ifc:relatingObject_IfcRelNests hil:HIL.HIL_flat.sent_VS_1 .
hil:IfcObjectDefinition_List_2436 a ifc:IfcObjectDefinition_List ;
list:hasContents hil:HIL.HIL_flat.sent_VS_1.port_a ;
list:hasNext hil:IfcObjectDefinition_List_2438 .
hil:IfcObjectDefinition_List_2438 list:hasContents hil:HIL.HIL_flat.sent_VS_1.port_b .

Verbindungen einer ausgewählten Komponente zu ihren Nachbarn
hil:IfcRelConnectsPorts_1973 a ifc:IfcRelConnectsPorts ;
ifc:globalId_IfcRoot hil:IfcGloballyUniqueId_1974 ;
ifc:relatedPort_IfcRelConnectsPorts hil:HIL.HIL_flat.sent_VS_1.port_b ;
ifc:relatingPort_IfcRelConnectsPorts hil:HIL.HIL_flat.dp_VS_1.port_b .

Zuordnung von Eigenschaften zu einer ausgewählten Komponente
hil:IfcRelDefinesByProperties_4842 a ifc:IfcRelDefinesByProperties ;
ifc:globalId_IfcRoot hil:IfcGloballyUniqueId_4843 ;
ifc:relatedObjects_IfcRelDefinesByProperties hil:HIL.HIL_flat.sent_VS_1 ;
ifc:relatingPropertyDefinition_IfcRelDefinesByProperties hil:IfcPropertySet_4840 .
hil:IfcPropertySet_4840 a ifc:IfcPropertySet ;
ifc:globalId_IfcRoot hil:IfcGloballyUniqueId_4841 ;
ifc:hasProperties_IfcPropertySet hil:IfcPropertySingleValue_4844,
    hil:IfcPropertySingleValue_4846,
    hil:IfcPropertySingleValue_4848 ;
ifc:name_IfcRoot hil:IfcLabel_3668 .
hil:IfcPropertySingleValue_4846 a ifc:IfcPropertySingleValue ;
ifc:name_IfcProperty hil:IfcIdentifier_3670 ;
ifc:nominalValue_IfcPropertySingleValue hil:IfcText_4847 .
hil:IfcIdentifier_3670 a ifc:IfcIdentifier ;
express:hasString "stringComment" .
hil:IfcText_4847 a ifc:IfcText ;
express:hasString "XLM10BTR01" .
```

Lis. 37 Auszug aus der HIL_ifc.ttl⁶⁵ (entspricht XYZ_ifc.ttl in Abb. 33 und Tab. 29)

5.2.3.3 RDF2IFC

Der entstandene IFC-KG wird nun mittels des RDF2IFC-Konverters in ein IFC-STEP-File überführt, dafür wurde die angepasste Version 1.1.0 des RDF2IFC-Transkriptor genutzt. Das entstehende IFC-STEP-File ist valide und kann ohne Fehlermeldungen z.B. mit dem KIT-Viewer geöffnet werden, besitzt jedoch noch keinerlei geometrische Repräsentation, dies wird im nächste Konvertierungsschritt behandelt.

⁶⁵ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2BIM/D_HIL/hil_ifc.ttl

Lis. 38 zeigt ausgewählte STEP-Entitäten, die zum oben bereits thematisierten Temperatursensor gehörig sind. Der Modelica-Identifikator `sent_VS_1` wird als `IfcRoot#Name` in die IFC-Datei überführt. Man erkennt weiterhin den erhaltenen `stringComment` und die Positionierung in der schematischen Ansicht. Dementsprechend sind die notwendigen Informationen für den folgenden Konvertierungsschritt, als auch für eine potentielle spätere Rückübersetzung (siehe Abschnitt 6.4.5) enthalten.

```
#5838= IFCSENSOR('2A5Fx9oTv9yBXMs_XbTRX',$, 'sent_VS_1', $$, $$, $$, .TEMPERATURESENSOR.);
#1535= IFCRELDEFINESBYPROPERTIES('0AWOFVDILCA8O$bdBxZ4Ty', $$, $$, (#5838), #1529);
#1529= IFCPROPERTYSET('3dSz8Qgan6OQ5y$YqNUVg$', $$, 'layout 2D', $$, (#1531, #1532));
#1531= IFCPROPERTYSINGLEVALUE('x coordinate', $$, IFCREAL(-150.00), $$);
#1532= IFCPROPERTYSINGLEVALUE('y coordinate', $$, IFCREAL(530.00), $$);
#4842= IFCRELDEFINESBYPROPERTIES('2gJh7WIHj48RdsLhKq2ZfQ', $$, $$, (#5838), #4840);
#4840= IFCPROPERTYSET('1ZqWSxr3HB0AsaAg$htkKP', $$, 'Modelica Infos', $$, (#4844, #4846, #4848));
#4844= IFCPROPERTYSINGLEVALUE('Modelica Class', $$, IFCTEXT('LibEAS.Sensors.TemperatureTwoPort_Display'), $$);
#4846= IFCPROPERTYSINGLEVALUE('stringComment', $$, IFCTEXT('XLM10BTR01'), $$);
#4848= IFCPROPERTYSINGLEVALUE('modifiction', $$, IFCTEXT('redeclare package Medium_gleich_Medium'), $$);
```

Lis. 38 Ausschnitt aus der HIL_F.ifc⁶⁶ (entspricht XYZ_F.ifc in Abb. 33 und Tab. 29)

5.2.3.4 Voluminizer

Für das Handling in IFC-Viewern und Autorentools ist es notwendig, die erzeugten IFC-Objekte mit einer geometrischen Repräsentation zu versehen. Dies realisiert der Konverter *Voluminizer*.

Zunächst wird die 2D-Repräsentation analog zur schematischen Anordnung im Simulationsmodell zugewiesen. Aufgrund der Beschränkungen vieler IFC-Viewer wird zusätzlich eine 3D-Geometrie generiert. Dabei wird entsprechend der Klasse und des *predefinedType* eines Objekts ein bestimmter Körper in einer festgelegten Farbe entsprechend Tab. 45 realisiert. Abb. 69 zeigt das Ergebnis.

Man erkennt, dass derzeit nur eine vereinfachte 2D-Repräsentation realisiert ist, dabei wird unabhängig von der Objekt-Klasse und -Typisierung immer dasselbe Symbol (Kreis) verwendet. Weiterhin werden alle semantischen Verbindungen durch eine gerade Verbindung visualisiert. Die Bezeichnung der Objekte wird als separate *IfcShapeRepresentation* mit dem *Identifier Annotation* (neben *Body* und *Footprint*) angelegt, damit können die verschiedenen Darstellungen mit einem geeigneten Ifc-Viewer (siehe Abb. 20) separat zu- und abgeschaltet werden.

⁶⁶ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2BIM/D_HIL/hil_F.ifc

5 Experimente

5.3 Vergleichende Auswertung zwischen Semantic und Procedural Translation

5.2.4 Auswertung

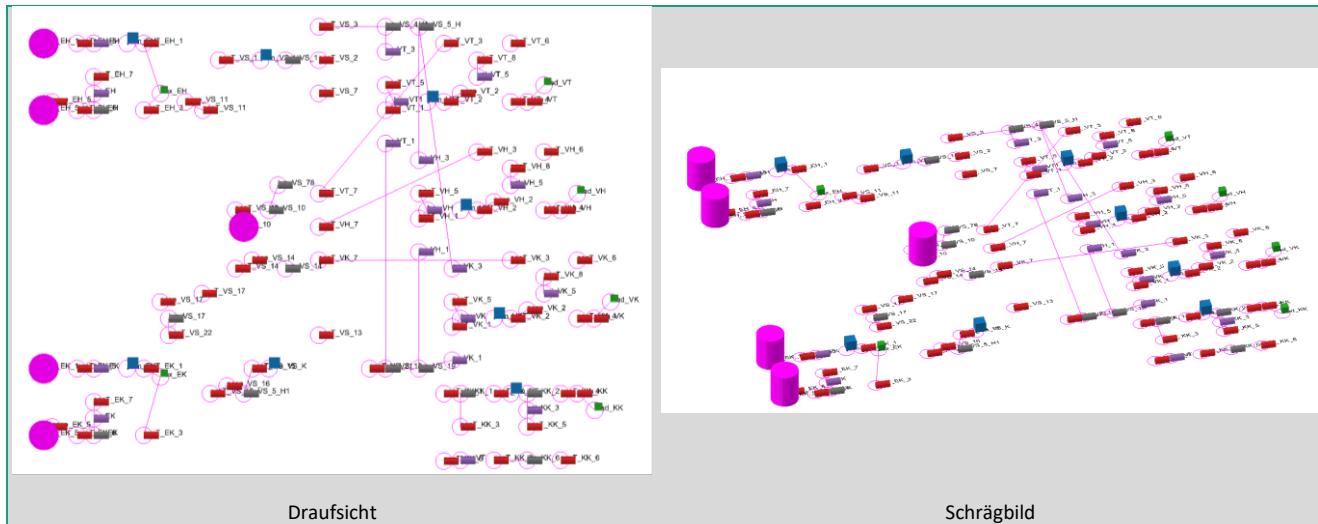


Abb. 69 angereichertes 3D-IFC-Modell des HIL-Prüfstands⁶⁷ (Screenshots aus dem KIT-Viewer (entspricht XYZ_F.ifc in Abb. 33 und Tab. 29)

5.2.4 Auswertung

Das entstandene IFC-Modell enthält die wesentlichen semantischen Informationen aus dem Modelica-Modell. Die Art des Bauteils in Modelica wird mittels der zugeordneten IFC-Klasse und des *predefinedType* abgebildet, zusätzlich wird eine *IfcProperty* mit dem Modelica-Klassennamen erzeugt. Die hydraulischen Verbindungen werden mit *IfcDistributionPorts* und *IfcRelConnects*-Relationen realisiert. Weiterhin sind der Modelica-Identifikator, sowie die von der Standardbelegung abweichenden Parameter im entstandenen IFC-Modell enthalten.

Dies ermöglicht die anschließende Weiterbearbeitung in Autorentools, wobei die wesentlichen Eigenschaften aus dem Simulationsmodell unmittelbar zur Verfügung stehen, insb. gilt dies für den Anlagengraph, der entscheidend für die hydraulische Funktionsweise des Systems ist.

5.3 Vergleichende Auswertung zwischen Semantic und Procedural Translation

Beide Varianten der Übersetzung (*Semantic Translation* und *Procedural Translation*) wurden implementiert und getestet. Es war jeweils möglich, die für die UseCases notwendigen Anforderungen zu realisieren.

Für die *Semantic Translation* ist es nicht, nötig Quellcode im engeren Sinne zu erstellen, stattdessen kommen marktverfügbare *Reasoner* z.B. im Protege-Editor zu Anwendung. Notwendig ist es jedoch, das entsprechende *Alignment* zu formulieren, welches dem *Reasoner* als Grundlage dient. Lis. 21 und Lis. 22 zeigen Beispiele für ein einzelnes *Alignment*. Die Trennung von Wissen (im KG) über, und Handlungsanweisungen (in Skripten/Code) zu Zuordnungen ist die Kernidee der *Semantic Translation*. Sie verursacht eine größere Anzahl von Artefakten (Dateien), welche administriert werden müssen.

Im Falle von 1:1-Zuordnungen ist die Syntax in OWL einfach und gut verständlich (*owl:equivalentClass*), dies gilt jedoch nicht mehr, wenn es sich um bedingte Zuordnungen handelt. Die dazu notwendigen OWL-Konstrukte sind umfangreich, schwerer verständlich und dadurch

⁶⁷ https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2BIM/D_HIL/hil_F_3D_2D.ifc

fehleranfällig (4.3.4.1, Lis. 21). Die alternative Darstellung mittels prozeduraler Anweisungen in Programmiersprachen sind vielen Nutzern geläufig und daher leichter verständlich.

Bei dem *Procedural Translator* handelt es sich um klassischen prozeduralen Quellcode. Lis. 30 zeigt den wesentlichen Teil des notwendigen Programmcodes. Man erkennt, dass dieser im Vergleich zu den Alignments relativ kurz und eingängig ist.

Neben dem Umfang des notwendigen Quellcodes ist die verwendete Syntax ein wesentlicher Unterschied zwischen beiden Lösungsansätzen. Während der prozedurale Quellcode in Lis. 30 einer Syntax folgt, die in vielen Programmiersprachen ähnlich ist, sind für die Alignments andere syntaktische Konstruktionen nötig. Diese haben ihren Ursprung in der formalen Logik und sind daher nur einem eingeschränkteren Anwenderkreis geläufig.

Der Vorteil der Semantic Translation ist, dass die Implementierung richtungsunabhängig ist. Das Alignment muss nur einmalig erstellt werden und kann dann für Übersetzungen in beide Richtungen verwendet werden.

Die in Abschnitt 3.11 thematisierten Laufzeitprobleme und hohen Anforderungen an den Arbeitsspeicher beim Semantic Reasoning konnten (bei Verwendung derselben Hardware) für die Procedural Translation nicht beobachtet werden.

Aufgrund der beschriebenen Nachteile der Semantic Translation und der technischen Beschränkung bei der Übersetzung in Hinblick auf *Data Type Properties* erscheint die *Procedural Translation* für die Übersetzungs-Use-Cases die geeigneter Wahl.

5 Experimente

5.3 Vergleichende Auswertung zwischen Semantic und Procedural Translation

5.2.4 Auswertung

6 Schlussfolgerungen und kritische Bewertung

Es erfolgt die Prüfung und Diskussion der in Kapitel 1 aufgestellten Thesen anhand der Erkenntnisse aus Implementierung (Kapitel 4) und Experimenten (Kapitel 5):

- Es ist möglich, aus IFC-Modellen automatisiert simulationsrelevante Informationen zu extrahieren.
- Das IFC-Format ist geeignet, um TGA-Schemata maschinenlesbar abzuspeichern.
- Aus einem solchen IFC-Schalschema lassen sich passende Modelica-Modelle mit geeigneter Modellstruktur und Vorparametrierung erzeugen.
- Wissensgraphen sind ein geeignetes Zwischenformat für die Übersetzung zwischen IFC und Modelica.
- Es wurde experimentell aufgezeigt, dass die Übersetzung in beiden Richtungen funktioniert.
- Die mit der Übersetzung einhergehenden Informationsverluste werden als unkritisch für den/die simulationsgestützten Anlagenentwurf bzw. -prüfung bewertet.

Anschließend werden die Grenzen der angewandten Methodik und Implementierung zusammengefasst: Insbesondere das Semantic Reasoning für die Übersetzung wirkt begrenzend.

Der Mehrwert der gefundenen Lösung besteht

- bei der Übersetzung von BIM2SIM in der Erweiterung auf die Anlagensimulation und
- bei der Übersetzung von SIM2BIM auf das erstmalige Erzeugen von Modellstrukturen in IFC.

Weitere Mehrwerte ergeben sich aus dem neu erzeugten Anknüpfungspunkt des Wissensgraphen – insbesondere im Bereich der Gebäudeautomation.

Im Ausblick werden darüber hinaus weitere Anknüpfungspunkte für den Wissensgraph und Nachnutzungsmöglichkeiten des MO-x-IFC-Toolsets aufgezeigt, welche vertiefte Untersuchungen und zusätzliche Implementierungen bedingen würden.

6.1 Prüfung und Diskussion der Thesen

In Abschnitt 5.1.3.2 wurde gezeigt, dass es möglich ist, aus Konstruktionsmodellen (wie in Abschnitt 1.1.2 und Abb. 6 und 8 eingeführt) automatisiert schema- und berechnungsrelevante Informationen zu extrahieren. Dafür werden nicht-strukturrelevante Objekte durch *Replacements* ersetzt. Wesentliche Voraussetzung dafür ist, dass im IFC-Modell Ports und ihre Verbindungen untereinander enthalten sind. Weitere Voraussetzungen wurden im Abschnitt 3.5.1 dokumentiert. Es wurden FLOSS-Tools entwickelt, um diese Voraussetzungen ggf. herzustellen (4.3.1). Die experimentelle Erprobung (5.1.3.1) zeigte, dass der dafür notwendige Aufwand in einem sinnvollen Verhältnis zur eingesparten Arbeitszeit steht. Alle Vorbereitungen konnten ohne kostenpflichtige Autorentools realisiert werden.

Es wurde gezeigt, dass die zugehörige zweidimensionale schematische Darstellung zusätzlich zur 3D-Geometrie in einem IFC-Modell abgespeichert werden kann. Diese wird als *IFC-Schalschema* bezeichnet und in Abschnitt 4.3.2 dargestellt. Vorteilhaft ist hier insbesondere, dass wesentliche Informationen nicht doppelt in verschiedenen Dokumenten gehalten werden, sondern es sich um eine *Single Source of Truth* handelt – Widersprüche sind somit ausgeschlossen und Konsistenzprüfungen unnötig. Ein wichtiger Aspekt ist die Anordnung der verbleibenden oder ersetzen Elemente in der zweidimensionalen schematischen Darstellung, da sich derartige schematische Darstellungen primär an menschliche Leser wenden und nur in zweiter Linie auch maschinenlesbar sein sollten.

Es wurde nachgewiesen, dass die *Semantic Translation* geeignet ist, aus dem *IFC Schalschema* eine valide Modellstruktur für ein Simulationsmodell zu erzeugen (5.1.3.4). Es ist jedoch nicht möglich die Parametrierung mit *Semantic Reasoning* zu realisieren, weshalb ein *Enhancement* des Graphen mittels SPARQL-Queries notwendig ist. Es wurde gezeigt, dass der notwendige Nacharbeitungsaufwand so gering ist, dass aus der Automatisierung ein wirtschaftlicher Vorteil entstehen kann.

Es wurde weiterhin gezeigt, dass es möglich ist, Modelica-Modelle als Wissensgraph abzubilden. Dafür wurden ausschließlich Aspekte, welche für die Übersetzung von und nach IFC relevant sind, behandelt. Dafür wurden der sog. MoTTL-Transcriptor (3.8.4) und die Modelica-Ontologie MoOnt (3.6) entwickelt. Damit ist es möglich, komplexe Abfragen an das Simulationsmodell zu stellen z.B. welche alternativen Implementierungen für ein bestimmtes Simulationsmodell in den eingebundenen Bibliotheken existieren. Weitere Anwendungsfälle werden im Ausblick (6.2) beschrieben.

Für die Übersetzung zwischen dem Mo-KG und dem IFC-KG wurde eine *Procedural Translation* implementiert. Es war damit möglich, die für ein IFC-Funktionsmodell notwendigen Informationen zu übertragen. Für die Übersetzung von Modelica nach IFC ist es weiterhin notwendig, eine 3D-Geometrie zu ergänzen (5.2.3.3). Dies erfolgt vereinfacht und stellt lediglich eine „Platzhalter-Geometrie“ dar, ermöglicht aber die Weiterbearbeitung in Autorentools.

Als Zwischenstände für die Übertragung wurden Wissensgraphen gewählt. Die für die Transkription in und aus diesem Format notwendigen Konverter wurden teilweise für die Modelica-Domäne selbst entwickelt. Für die IFC-Domäne existierten sie bereits.

Die Übersetzung zwischen den Domänen Modelica und IFC geht für beide Richtungen mit Informationsverlust einher. Dies ist jedoch kein Fehler der Implementierung, sondern vielmehr eine notwendige Folge aus den unterschiedlichen beteiligten Modellen, die die Realität jeweils unter verschiedenen Aspekten abbilden. Es wurden dafür die Begriffe Konstruktions- und Funktionsmodell eingeführt, die signifikant unterschiedliches Wissen enthalten. IFC- und

Modelica-Modelle realisieren oftmals diese beiden Konzepte und müssen daher unterschiedliches Wissen enthalten. Der Fokus der Arbeit lag darauf, die Schnittmenge des Wissens zwischen beiden Domänen zu übertragen. Es wurde experimentell nachgewiesen, dass dies in einem Maß möglich ist, dass es zur Einsparung von Arbeitszeit führt.

6.2 Limitationen

6.2.1 Grenzen der Methodik

Semantic Reasoning als technische Grundlage der *Semantic Translation* funktioniert richtungsunabhängig und bietet somit theoretisch die Möglichkeit, automatisch eine bidirektionale Übersetzung zu ermöglichen. Die Nachteile des Semantic Reasoning wurden in Abschnitt 5.3 zusammengefasst: Von zentraler Bedeutung ist dabei die Beschränkung bzgl. des Schlussfolgerns von DataTypeProperties. Daher kann eine automatische, hinreichend vollständige bidirektionale Übersetzung nicht erreicht werden. Die Procedural Translation ist der sinnvollere Lösungsansatz, muss jedoch für beide Richtungen separat implementiert werden.

6.2.2 Grenzen der vorliegenden Implementierung

Der vorliegende Workflow funktioniert für IFC-Modelle von Heizungs- und Kälteanlagen. Die Architekturdomäne (Bauteile) sowie andere Arten von Anlagen insb. Lüftung und Elektrotechnik wurden nicht betrachtet.

Die wesentliche Anforderung an die zu verwendenden IFC-Modelle ist das Vorhandensein eines Anlagengraphen im Modell. Alle anderen in Abschnitt 3.5.1 zusammengetragenen Anforderungen können u. A. mit Hilfe der bereitgestellten Skripte mit überschaubarem Aufwand, auch ohne Verwendung von Autorentools, hergestellt werden.

Der experimentelle Nachweis wurde anhand eines Praxismodells eines mittelgroßen Gebäudes erbracht. Dafür wurde ein Standard-Desktoprechner aus dem Jahr 2019 benutzt. Lediglich beim Schritt Semantic Translation wurden nennenswerte Rechenzeiten beobachtet, bei der alternativen Implementierung als Procedural Translation waren die Rechenzeiten unterhalb der Beobachtungsschwelle.

Die Testmodelle für die beiden Experimente in Kapitel 5 bestanden jeweils aus etwa 200 Komponenten, was in etwa die Grenze für die Handhabung eines unverschachtelten Modells darstellt. Aufgrund der Laufzeitcharakteristik der entwickelten Konverter sind auch für deutlich größere Modelle akzeptable Umwandlungszeiten zu erwarten.

6.3 Mehrwert gegenüber bestehenden Lösungen

6.3.1 Übersetzung BIM2SIM

Im Gegensatz zu den in Abschnitt 1.4.1 vorgestellten Arbeiten wurde in dieser Arbeit nicht die Gebäude-, sondern die Anlagensimulation adressiert. Dieses *Model of Computation* wurde von den meisten anderen Arbeiten, welche sich mit der Übertragung von IFC-Informationen nach Modelica befassen, gar nicht oder nur am Rande thematisiert. Der Funktionsumfang des vorliegenden Toolsets ist demgegenüber deutlich erweitert.

Die existierenden Vorarbeiten welche die Anlagensimulation adressieren, verwenden ein herstellerneutrales Modellformat namens *Simmodel*. Dieses muss in einem unabhängigen Tool (nicht in der IFC-Domäne, aber auch nicht in der Modelica-Domäne) mit Informationen, welche die Simulation zusätzlich zu den Informationen im IFC-Modell benötigt, angereichert werden. Ein solches „dritten“ Tool ist für den hier vorgestellten Workflow nicht notwendig. Informationen

werden stattdessen ausschließlich in den ohnehin beteiligten Tools (IFC-Viewer oder Autorentool und Simulationsumgebung) eingepflegt.

6.3.2 Übersetzung SIM2BIM

Die Übersetzung von Modelica zu IFC wurde von den meisten Autoren nur in Hinblick auf die Übertragung von Parametern betrachtet. Die vorliegende Arbeit beschäftigt sich davon abweichend mit der Erstellung der Modellstruktur in IFC basierend auf Modelica-Informationen. Dies ermöglicht den simulationsbasierten Anlagenentwurf. Es bildet weiterhin die Grundlage für die Realisierung einer Round-Trip-Lösung, wie sie in Abschnitt 6.4.4 skizziert wird.

6.4 Ausblick

6.4.1 Anknüpfungspunkt Wissensgraph

Als Zwischenschritte der bidirektionalen Übersetzung wurden neben den nativen Dateiformaten ausschließlich Wissensgraphen verwendet. Das notwendige Vokabular (MoOnt) und die zugehörigen Transkriptoren (MoTTL-Transcriptor, TTL2MO) für die Modelica-Domäne wurden entwickelt.

Der MoTTL-Transcriptor ermöglicht die automatisierte Übertragung von Modelica-Modellen in Wissensgraphen. Damit besteht ein wesentlicher Unterschied zu den in Abschnitt 1.4.3 beschriebenen Arbeiten, die sich mit der Abbildung von Modelica-Modellen als Wissensgraphen beschäftigen, dies jedoch manuell versucht haben. Der wesentliche Vorteil des automatisierten Ansatzes besteht darin, dass er große Datenmengen verarbeiten kann und dass er wiederholt ausgeführt werden kann, z.B. wenn eine der Bibliotheken oder die Modelle aktualisiert werden.

Der Mehrwert der Lösung entsteht durch die Interoperabilität der Darstellung als Wissensgraph. Es sind zahlreiche weitere Anwendungen der entwickelten Komponenten denkbar. Im Folgenden werden weitere UseCases beschrieben, welche in dieser Arbeit nicht experimentell untersucht wurden.

6.4.1.1 Planung Gebäudeautomation

Es wurde in Abschnitt 1.1.1 bereits auf die oftmals bestehende „Performance Gap“ zwischen geplantem Energiebedarf und tatsächlichem Energieverbrauch hingewiesen. Eine wesentliche Rolle bei der Adressierung dieses Umsetzungsdefizits spielt die Programmierung und Inbetriebnahme der Gebäudeautomation (Cozza u. a. 2021; van Dronkelaar u. a. 2016). Für eine energieeffiziente Betriebsweise des Gebäudes ist nicht nur ein gutes Anlagenkonzept wichtig, sondern auch die dazu passende Regelung (Eckstädt und Bräunig 2024). Die Fachplanung Gebäudeautomation (GA) befasst sich damit und es ist somit wichtig, Informationsverluste an der Schnittstelle zwischen TGA- und GA-Fachplanung zu vermeiden.

Für die Kostenschätzung des Gewerks Gebäudeautomation ist regelmäßig die Anzahl der Datenpunkte entscheidend. Diese kann basierend auf dem Simulationsmodell der Anlage sehr genau ermittelt werden, denn die Anzahl der Sensoren und Aktoren kann dem Modell entnommen werden. Die dafür geltenden Voraussetzungen des Simulationsmodells und eine beispielhafte Anwendung wurden in (Eckstädt, Menzel, u. a. 2023) beschrieben.

6.4.1.2 Programmierung der Gebäudeautomation

Planung und Realisierung der Gebäudeautomation knüpfen dabei an das Funktionsmodell der TGA-Anlage an. Herkömmlich funktioniert die Kommunikation zwischen beiden Gewerken fast ausschließlich über die TGA-Schalschemata (für die einzelnen Untergewerke Heizung, Lüftung, Kühlung). Jedoch ist dieses nicht ausdrucksstark genug, um insb. die Detailinformationen zur Regelung zu tragen. Diese werden üblicherweise als Freitext-Anforderung an den

Anlagenprogrammierung kommuniziert (als Anlage zum oder Ergänzung auf den TGA-Schemata). Freitexte sind jedoch nicht gut geeignet, um (komplexe) Regelalgorithmen und ihre genaue Parametrierung zu beschreiben. Eine direkte Übergabe von Detailinformationen aus dem Simulationsmodell kann diesen Mangel ggf. beheben.

Mit der Control Description Language (CDL) wurde von (Wetter, Grahovac, und Hu 2018) ein Vorschlag zur herstellerneutralen Beschreibung von Regelungsinformationen vorgeschlagen. CDL basiert auf Modelica und knüpft somit ideal an die in dieser Arbeit beschriebenen UseCases an. Das IFC-Format hat bzgl. Regelungstechnik zum derzeit aktuellen Stand IFC 4.3 nur einen eingeschränkten Funktionsumfang und erscheint weniger geeignet, die notwendigen Informationen abzuspeichern: Es ist lediglich möglich auszudrücken, welcher Regler auf welchen Aktor wirkt, jedoch nicht die dahinterliegende Regelungslogik. Der in Abschnitt 1.2.3.1 und 6.4.6 beschriebene MultiModel-Ansatz ist dafür ggf. eine geeignete Lösung.

6.4.1.3 Inbetriebnahme und Betrieb von TGA und GA

Eine maschinenlesbare Übergabe der Regelungsinformationen kann insb. in der Inbetriebnahmephase gewinnbringend genutzt werden. Es ist damit möglich, automatisiert Testfälle zu definieren und daraus Inbetriebnahmeszenarien abzuleiten.

Die Gebäudeautomation ermöglicht mit ihren aufzeichbaren Messwerten das Monitoring des Gebäudes, dabei können insb. fehlerhafte Parametrierungen und Programmierung erkannt werden. Beides führt, wenn es unerkannt bleibt, möglicherweise zu erhöhtem Anlagenverschleiß (z.B. „schwingende“ Regelkreise und daraus resultierend stark schwankende Ventilhübe) und/oder schlechter Effizienz der Anlagen (z.B. falsch gewählte Grenztemperaturen zwischen freier und aktiver Kühlung oder fehlende Sperrung einer Kühlung durch Betonkernaktivierung gegen eine Heizung mit Heizkörper). In der Betriebsphase kann eine kombinierte Gebäude- und Anlagensimulation als kontinuierlicher Benchmark dienen und somit helfen die „Performance Gap“ zu verringern.

6.4.1.4 SIM2BRICK

Alternativ zu einer Übersetzung von Modelica in IFC ist eine Übersetzung von Modelica in BRICK denkbar. Die BRICK-Ontologie dient der Beschreibung von Anlagen. Sie wurde von (Balaji u. a. 2018) vorgestellt, hat eine rege Community und einige namhafte Industriepartner. Dafür müsste entweder ein alternatives Alignment zur Nutzung mit dem *Semantic Translator* erstellt werden oder ein angepasster *Procedural Translator*. Eine Transkription auf BIM-Seite ist nicht notwendig, da es sich bei BRICK nativ um einen Wissensgraphen handelt. Es wäre damit denkbar, den Entwurf eines Anlagenkonzepts in der Simulationsumgebung (z.B. OpenModelica) auszuführen und anschließend einen BRICK-Graphen daraus abzuleiten.

6.4.1.5 Analyse von Simulationsbibliotheken

In Abschnitt 3.7 wurde die Überführung von Simulationsbibliotheken in Wissensgraphen beschrieben. Dies ist eine notwendige Voraussetzung für die Übersetzung von Instanz-Modellen, wie sie in Kapitel 5 experimentell gezeigt wurde. Die Wissensgraphen der Bibliotheken können jedoch zur Analyse der Simulationsbibliotheken genutzt werden, beispielsweise für die Ermittlung aller Modelle, welche eine bestimmtes abstraktes Modell (*partial*) implementieren. Derartige Beispiele wurden in (Eckstädt, Menzel, u. a. 2023) gezeigt.

6.4.2 Übertragbarkeit auf andere Simulationsmodelle

Eine Übertragbarkeit auf Simulationsmodelle jenseits der in Abschnitt 3.8.4.3 beschriebenen, ist im Falle der Semantic Translation einfach durch eine Erweiterung des Alignments möglich. Im Falle der Procedural Translation ist dafür eine Anpassung der „Dictionaries“ im Quelltext des *Procedural*

Translator (4.3.7) für die Übersetzung in beide Richtungen separat notwendig. Um eine adäquate Darstellung im IFC-Modell zu erreichen, sollte dann auch der *Voluminizer* angepasst werden.

Denkbare sinnvolle Erweiterungen wären z.B. solche auf Raummodelle, Lüftungskomponenten und standardisierte Regelungskomponenten gemäß CDL. Vorarbeiten hinsichtlich der Lüftungskomponenten wurden bei (Manotas 2021) dargestellt.

6.4.3 Anwendung für die gekoppelte Simulation

Für einige der Anwendungsfälle in der simulationsgestützten Planung ist eine echte Kopplung zwischen Gebäude- und Anlagensimulation notwendig (siehe Abb. 12). Im Gegensatz zur „seriellen Kopplung“ ist hierfür zu jedem Simulationszeitschritt (entspricht z.B. 1h oder 1min in Echtzeit) ein Austausch der Simulationsergebnisse zwischen Gebäude- und Anlagensimulation notwendig. Dafür ist es entweder notwendig, beide Simulationen in derselben Umgebung⁶⁸ auszuführen oder eine Framework für die Simulatorkopplung zu nutzen. In (Eckstädt, Huang, u. a. 2023) wurde dies näher beschrieben.

Das „Functional Mockup Interface“ (FMI) ist ein solches Framework für die Simulatorkopplung. Es dient der gekoppelten Simulation mehrerer „Functional Mockup Units“ (FMU) in einem sog. FMU-Master. FMUs sind – ähnlich wie Modelica-Modelle – durch definierte Schnittstellen und Parameter charakterisiert („Shape“ der FMU). Eine gekoppelte Simulation definiert sich über die enthaltenen FMUs und deren Verbindungen. Dafür könnte – analog zu MoOnt – die „FMUOn“ von (Mitterhofer 2018) verwendet werden. Dies könnte wie in Abschnitt 3.2 und bei (Zeb & Kortelainen, 2017) bereits beschrieben Mehrwerte in Hinblick auf Langzeitverfügbarkeit der Informationen bringen.

6.4.4 Verbesserungsmöglichkeiten in Hinblick auf die Implementierung

Schema

Die Darstellung des IFC-Schalschemas könnte verbessert werden, indem Symbole – statt der derzeitigen Kreis- und Strichgeometrie – sowie unterschiedliche Farben für verschiedene *IfcSystem* benutzt werden. Weiterhin würde eine etagenweise Anordnung und eine orthogonale Linienführung die Darstellungsweise der üblichen Darstellung (Abb. 50) angelehen.

Weiterhin wäre es denkbar, Exporte des generierten Schemas in ein 2D-CAD-Format vorzunehmen, da diese derzeit noch vielfach Stand der Technik sind und somit der simulationsbasierte Anlagenentwurf auch Planern zugänglich würde, die noch nicht mit der BIM-Methodik arbeiten.

Parametrierung

Die Eigenschaften der Objekte werden bisher für beide Übersetzungsrichtungen von den Konvertern nur rudimentär übertragen. Die in Abschnitt 1.4.1 und 1.4.2 thematisierten Forschungsansätze aufgreifend, könnten die vorliegenden Konverter um diese Funktionalität ergänzt werden.

insb. für die Komponenten, welche in hoher Stückzahl vorkommen, wie z.B. Heizkörper und Umluftkühler, würde diese Erweiterung erheblich Arbeitszeit einsparen. Dabei handelt es sich oftmals um die raumzugeordneten Übergabeeinheiten. Eine Voraussetzung für eine automatisierte Übertragung der Eigenschaften ist eine Festlegung unter welchem Namen und an welcher Stelle (*IfcPropertySet*) diese Informationen im BIM-Modell zu finden sind. Dies kann projektspezifisch mit Auftraggeber-Informations-Anforderungen oder allgemeinverbindlicher festgelegt werden. Aktuell wird in den Richtlinienprojekte VDI 6070 Blatt 2ff und VDI 2552 Blatt 11.9 daran gearbeitet, dies zu standardisieren.

⁶⁸ Dieser Ansatz wird z.B. von IDA-ICE, TRNSYS-TUD und Energy Plus verfolgt. Er kann auch mit Modelica realisiert werden.

6.4.5 Auswertung in Hinblick auf Round-Trip

Mit den UC1 BIM2SIM und UC2 SIM2BIM wurde die erstmalige Erzeugung eines BIM-Modells aus einem Simulationsmodell und die erstmalige Erzeugung eines Simulationsmodells aus einem BIM-Modell beschrieben. Aus der Kombination beider UseCases ergibt sich der *Round-Trip*, welcher Änderungen in einem der beiden Modelle jeweils auf das andere zurückspiegelt. Dies können z.B. Aktualisierungen am Erzeugermodell sein, die in der Simulationssoftware durchgeführt und nunmehr in das BIM-Modell zurück übertragen werden sollen. Die Notwendigkeit dafür ergibt sich aus den internen Iterationen zwischen 3D-Konstruktionsmodell und Berechnungsmodell wie sie in Abb. 4 dargestellt sind.

Die praktische Relevanz des *Round-Trip* ergibt sich entsprechend der Ausführungen in Abschnitt 1.1.2 aus der iterativen Weiterentwicklung von Funktions- und Konstruktionsmodell, die in Abb. 6 und Abb. 8 hervorgehoben ist.

Für die serielle Übersetzung von einem BIM-Modell zu einem Simulationsmodell und zurück bzw. ausgehend von einem Simulationsmodell zu einem BIM-Modell („*Round-Trip*“) und zurück ist es notwendig, eine eindeutige Zuordnung (siehe Abschnitt 3.8.4.2) zwischen beiden Daten-Schemata vornehmen zu können. Die in dieser Arbeit implementierten Mapping-Tabellen (Tab. 38) zeigen jedoch, dass die nicht gegeben ist. Es gibt sowohl den Fall, dass eine Entität aus Modelica aus unterschiedlichen IFC-Klassen entstanden sein kann, als auch den umgekehrten Fall, dass eine bestimmte Klasse-Type-Kombination in IFC aus unterschiedlichen Komponenten-Klassen in Modelica enstanden sein kann. Bei einer Rückübersetzung in die Ursprungsdomäne wäre die Zuordnung dann uneindeutig.

Für die Realisierung des *Round-Trip* wären daher in beiden tangierten Domänen (IFC und Modelica) zusätzliche Softwarebausteine – im Folgenden als *Updater* bezeichnet – notwendig. Diese führen zunächst eine Fallunterscheidung durch, um festzustellen, ob ein Modell erstmals erstellt werden soll oder ob es sich um den Rückweg zu einem bestehenden Modell handelt. In ersterem Fall wird die Vorgehensweise, wie sie in dieser Arbeit beschrieben wurde, angewendet (SIM2BIM oder BIM2SIM). In letzterem Fall muss ein Abgleich zwischen Ursprungsmodell und neuer Version erfolgen. Abb. 70 zeigt, wie sich das in den bisher beschriebenen Workflow einordnen würde. Es sind dabei drei Fälle zu unterscheiden in Abhängigkeit davon, welches Modell als „Master“ dienen soll: In der IFC-Domäne sind sowohl das Konstruktions-, als auch das IFC-Funktionsmodell Kandidaten, in der Modelica-Domäne ist nur das Simulationsmodell ein Kandidat. Das Strukturmodell (*XYZ_raw.mo*) als Zwischenschritt bei der Erstellung eines Simulationsmodells im BIM2SIM-UseCase enthält per Definition keine zusätzlichen Informationen.

Analog zu den aus der Softwareentwicklung bekannten Begriffen aus der Versionsverwaltung kann das Ursprungsmodell dabei als *master* betrachtet werden. Die geänderte Version *commit*, muss mit dem *master* zusammengeführt (*merge*) werden, was ggf. eine komplexe Aktivität erfordert.

Die Vorgehensweise zur Implementierung wäre dreiteilig vorzusehen und wird beispielhaft für den Fall „Update Funktionsmodell“ aus Abb. 70 skizziert:

1. Neues IFC-Funktionsmodell (*commit*) erzeugen aus Modelica (entspricht UC SIM2BIM)
2. Vergleich von altem IFC-Funktionsmodell (*master*) und neuem IFC-Funktionsmodell (*commit*) in Hinblick auf Bestand/Entfall/Ergänzung von Komponenten und Eigenschaften (Attribute/Properties), für Eigenschaften sind außerdem inhaltliche Änderungen denkbar. Die Unterschiede (DIFF) zwischen beiden Modellen müssen zusammengetragen werden.
3. Diese Unterschiede (DIFF) müssen dann auf das IFC-Funktionsmodell (*master*) angewendet werden.

6 Schlussfolgerungen und kritische Bewertung

6.4 Ausblick

6.4.6 Einordnung in dDTw-Konzept und Softwarearchitektur von iECO

Wissenschaftliche Vorarbeiten in diesem Bereich wurden bereits erbracht (Esser, Vilgertshofer, und Borrmann 2023; 2022; 2021; Liu, Gao, und Gu 2023). Die Voraussetzungen für den *Round-Trip* wurden mit den Implementierungen dieser Arbeit geschaffen, indem eindeutige Identifikatoren der Modellkomponenten mitgeführt bzw. bei Neuerstellung generiert wurden.

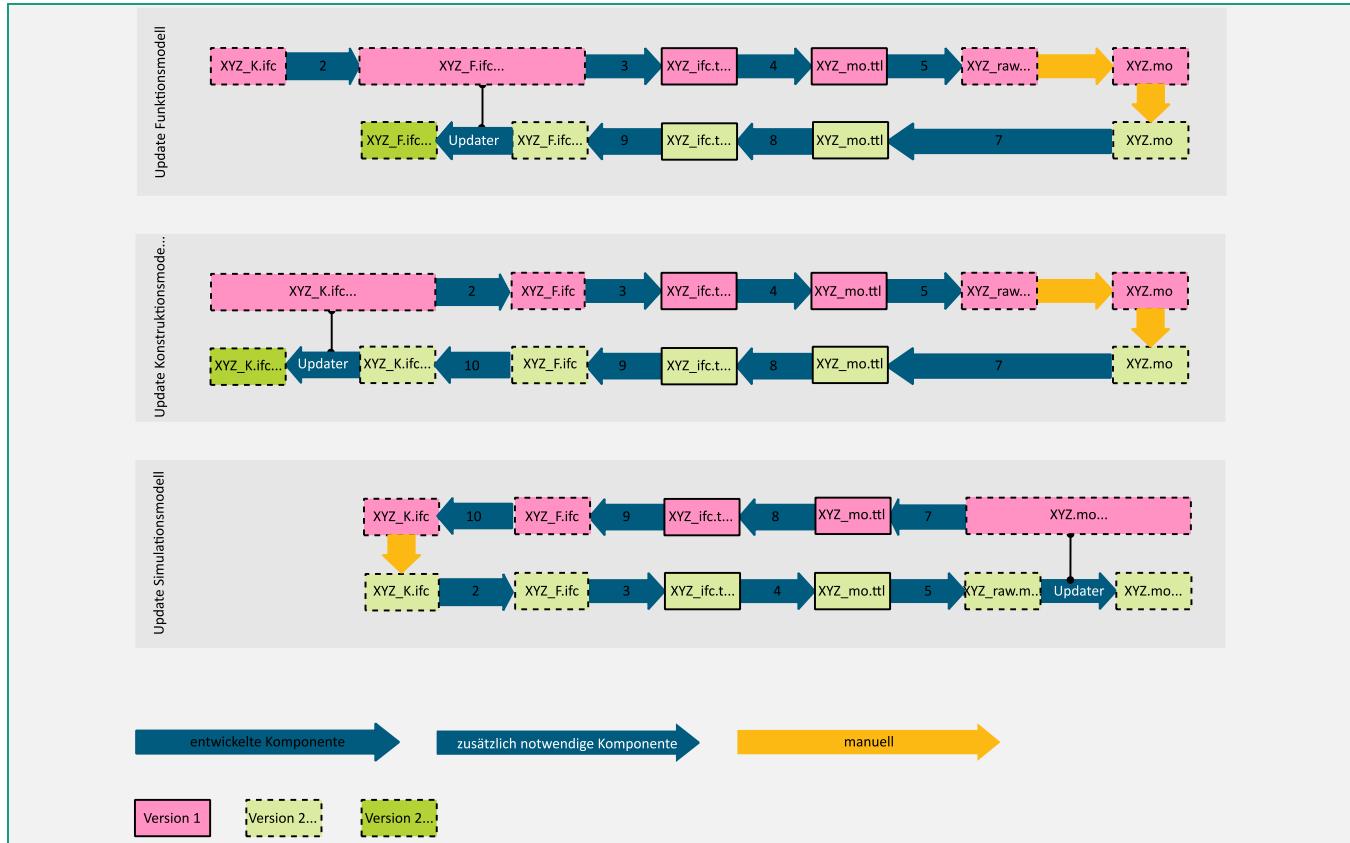


Abb. 70 Softwarekomponenten für Round-Trip BIM2SIM2BIM

6.4.6 Einordnung in dDTw-Konzept und Softwarearchitektur von iECO

Im Projekt iECO wurde eine Umsetzung des Konzepts „distributed digital twin (dDTw)“ mit Hilfe des GAIA-X Frameworks konzipiert (Strnadl u. a. 2024; Strnadl und Schöning 2023). Die Vorteile dieses Konzepts liegen in der Kombination aus Datensouveränität beim jeweiligen Modellautor und der dennoch bestehenden Interoperabilität.

Abb. 71 zeigt das Architekturdiagramm für diese Umsetzung. Man erkennt, dass die beiden Prozessbeteiligten „Participant A“ und „Participant B“ jeweils ein eigenes Modell in beliebigem Format haben (Nr. 7). Dazu kann jeweils ein semantisches Pendant als Wissensgraph (Nr. 8 „Ontology“) existieren. Diese Wissensgraphen können im „dDTw Repository“ (Nr. 1) enthalten oder verlinkt sein. Zwingender Bestandteil des dDTw Repository ist hingegen das sog. Linkmodell (Nr. 6). Dies kann ggf. durch einen dritten „Participant V“ erstellt werden.

Die Anwendungsfälle in dieser Arbeit beschränken sich auf zwei Beteiligte – den Planungsingenieur für TGA und den Simulationsingenieur (Participant A und B in Abb. 71). Es sind im Wesentlichen zwei Modelle beteiligt: Das Konstruktions- (Modell A) und Funktionsmodell (Modell B), beide können unterteilt werden in die anlagen- und raumseitige Darstellung, wie in Abb. 6 und Abb. 8 dargestellt. In dieser Arbeit wurden keine Linkmodelle gemäß Abb. 71 erstellt, jedoch wäre dies mit der bereits implementierten Fachlogik einfach möglich. Die Zuordnung der Modellentitäten im

Konstruktions- und Funktionsmodell war Gegenstand der Betrachtungen insb. in den Abschnitten 3.8.4.3 und 3.11.

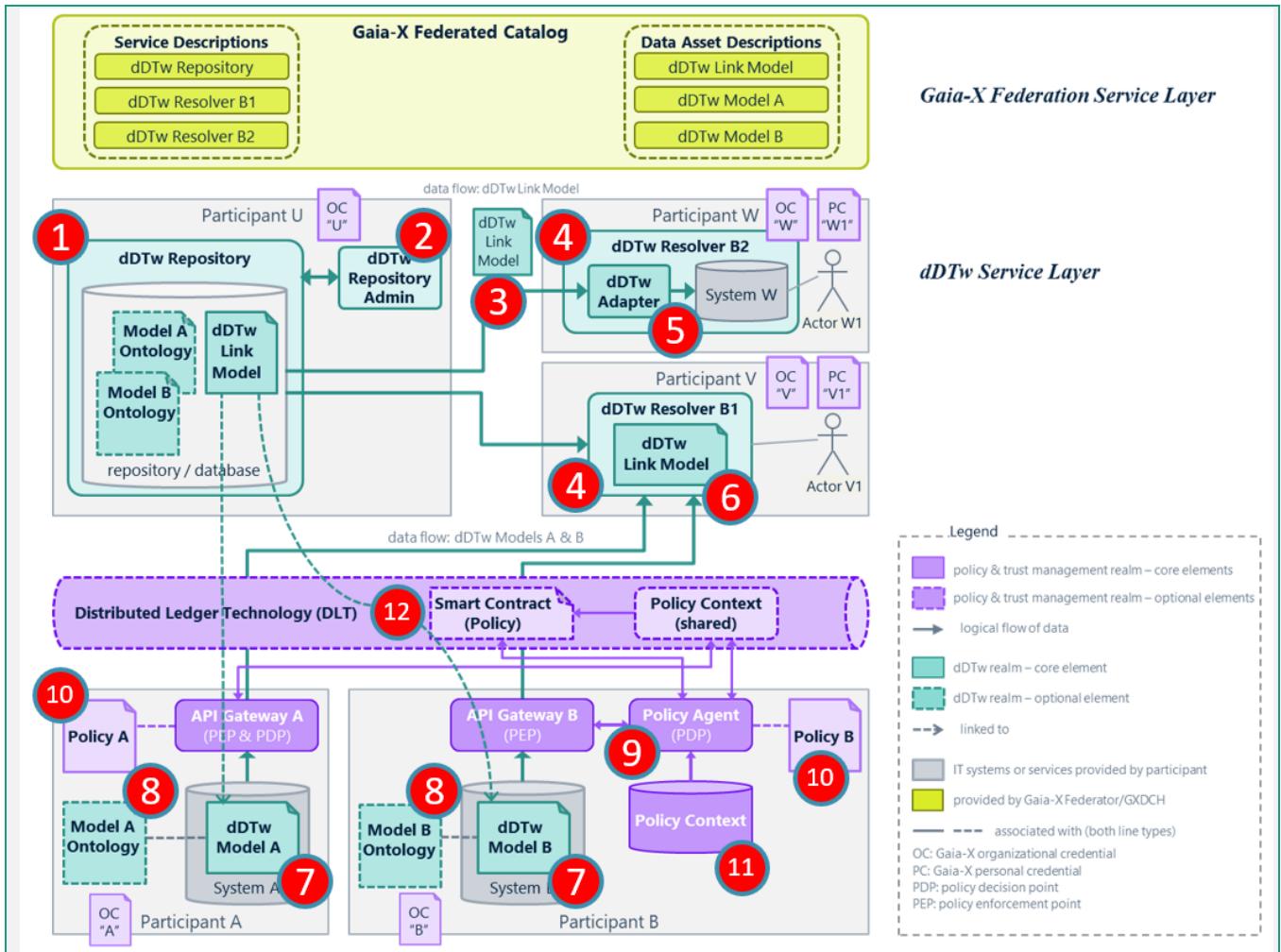


Abb. 71 Architektur dDTw auf GAIA-X (Bildquelle: Michael Polter nach (Strnadl u. a. 2024))

Quellenverzeichnisse und Anhang

Verzeichnis verwendeter Software

Die für die Erstellung dieser Arbeit verwendete Software/Tools sind in den folgenden Tabellen aufgelistet.

Tab. 54 FLOSS-Software

Bezeichnung	Version	Kurzbeschreibung	Website	Autor/Herausgeber
AixLib	1.3.0	Modelica Bibliothek für Gebäude- und Anlagensimulation	github.com/RWTH-EBC/AixLib	RWTH Aachen EBC (Maier, Laura u. a. 2023)
ANTLR		Parser Generator	www.antlr.org	(Parr 2013)
Apache Jena	-	Java Framework für Semantic Web und Linked Data	jena.apache.org	Apache Foundation
KIT-Viewer (früher: FZK-Viewer)	7.1	IFC-Viewer	www.iai.kit.edu/1302.php	KIT Karlsruhe
Gephi	0.10.1	Visualisierungstool für Graphen	gephi.org	(Bastian, Heymann, und Jacomy 2009)
Graphviz		Graph-Layouting Tool	graphviz.org	
Ifcopenshell		Python-Bibliothek für Handling von IFC	ifcopenshell.org	
IfctoRDF		Konverter von IFC-STEP zu IFC-OWL	github.com/pipauwel/IFCToRDF	
IfcOWL-to-IfcSTEP		Konverter von IFC-OWL zu IFC-STEP	github.com/BenzlyZhang/IfcSTEP-to-IfcOWL-converters	
MBL Modelica Building Library	9.0.0	Modelica-Bibliothek für Gebäude- und Anlagensimulation	simulationresearch.lbl.gov/modelica	LBLN Berkeley (Wetter u. a. 2014)
MLS Modelica Language Specification		Sprachspezifikation Modelica	specification.modelica.org/master	Modelica Association
Modelica-Builder		Python-Bibliothek zum Handling von Modelica Modellen	github.com/urbanopt/modelica-builder	
MSL Modelica Standard Library	4.0.0	Modelica-Bibliothek für Standard-Komponenten	github.com/modelica/ModelicaStandardLibrary	Modelica Association
Networkx		Python-Bibliothek für Handling von Graphen	networkx.org	
OpenModelica		Modelica-Simulator	openmodelica.org	Open Modelica Consortium (Fritzson u. a. 2020)
OWLReady2	0.42	Python-Bibliothek für Handling von OWL	owlready2.readthedocs.io/en/v0.42	(Lamy 2017)
Protege		Ontologieeditor	protege.stanford.edu	(Musen 2015)
RDF		Ressource Description Framework	www.w3.org/RDF	W3C
RDFLib	7.0.0	Python-Bibliothek zur Arbeit mit RDF	rdflib.readthedocs.io/en/stable/index.html	
Solibri Optimizer	2.1.4	Werkzeug zur Verringerung der Dateigröße	www.solibri.com/news/solibri-ifc-optimizer-2-1-4-release-notes	Solibri
SPARQL	1.1	Abfragesprache für RDF	www.w3.org/TR/sparql11-query www.w3.org/TR/sparql11-update	W3C
Terse RDF Triple Languge	1.1	Syntax für RDF	www.w3.org/TR/turtle	W3C
Wsm Wolfram System Modeller		Modelica Ontologie	www.sprint-iot.eu/Wolfram-Modelica-ontology.zip	SPRINT Konsortium
Draw.io		Zeichentool	www.drawio.com	
Zotero	6.0.36	Literaturverwaltung	www.zotero.org	

Tab. 55 kommerzielle Software

Bezeichnung	Kurzbeschreibung	Website
Dymola	Modelica-Simulator	www.3ds.com/de/produkte-und-services/catia/produkte/dymola/
simpleBIM	IFC-Viewer	simplebim.com/
Microsoft Office	Textverarbeitung, Tabellenkalkulation	www.office.com/

Verzeichnis der zitierten Normen

Tab. 56 zitierte Normen

Kürzel	Bezeichnung	Aktuelle Ausgabe	Nähere Informationen
DIN 18290	Verlinkter BIM-Datenaustausch von Bauwerksinformationsmodellen mit weiteren Fachmodellen	Reihe	Tab. 1
DIN 18290-1	Teil 1: Verlinkter Datenaustausch mehrerer Fachmodelle beim Building Information Modeling	2023-11	Tab. 1
DIN EN 12831-1	Energetische Bewertung von Gebäuden - Verfahren zur Berechnung der Norm-Heizlast - Teil 1: Raumheizlast	2017-09	
DIN EN ISO 12006	Hochbau Organisation des Austausches von Informationen über die Durchführung von Hoch- und Tiefbauten	Reihe	Tab. 2
DIN EN ISO 16757	Datenstrukturen für elektronische Produktkataloge der Technischen Gebäudeausrüstung	Reihe	Tab. 1
DIN EN ISO 17412	Bauwerksinformationsmodellierung – Informationsbedarfstiefe	Reihe	Tab. 2
DIN EN ISO 19650	Organisation und Digitalisierung von Informationen zu Bauwerken und Ingenieurleistungen, einschließlich Bauwerksinformationsmodellierung (BIM) - Informationsmanagement mit BIM	Reihe	
DIN EN ISO 19650-1	Teil 1: Begriffe und Grundsätze	2019-08	
DIN EN ISO 21597	Informationscontainer zur Datenübergabe - Austausch-Spezifikation	Reihe	Tab. 1
DIN EN ISO 21597-1	Teil 1: Container	2021-07	Tab. 1
DIN EN ISO 23386	Bauwerksinformationsmodellierung und andere digitale Prozesse im Bauwesen – Methodik zur Beschreibung, Erstellung und Pflege von Merkmalen in miteinander verbundenen Datenkatalogen	2020-11	Tab. 2
DIN EN ISO 23387	Bauwerksinformationsmodellierung (BIM) - Datenvorlagen für Bauobjekte während des Lebenszyklus eines baulichen Vermögensgegenstandes - Konzepte und Grundsätze	2020-12	Tab. 2
DIN EN ISO 29481	Bauwerksinformationsmodelle - Handbuch der Informationslieferungen	Reihe	Tab. 2
ISO 10303	Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch	Reihe	
ISO 10303-21	Teil 21: Implementierungsmethoden: Klartext-Kodierung der Austauschstruktur	2016-03	
ISO 12911	Organisation und Digitalisierung von Informationen zu Bauwerken und Ingenieurleistungen, einschließlich Bauwerksinformationsmodellierung (BIM) - Rahmen für die Spezifikation der Implementierung von Bauwerksinformationsmodellierung (BIM)	2023-02	Tab. 2
ISO 16739-1	Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries - Part 1: Data schema	2024 (IFC 4.3) bzw. 2018 (IFC 4.0)	Abschnitt 2.1
ISO 22263	Organization of information about construction works — Framework for management of project information	2008-01	
VDI 2078	Berechnung der thermischen Lasten und Raumtemperaturen (Auslegung Kühllast und Jahressimulation)	2015-06	
VDI 2552	Building Information Modeling (BIM)	Reihe	Tab. 2
VDI 2552-11	BIM - Informationsaustauschanforderungen	Reihe	Tab. 2
VDI 2552-11.9	BIM - Informationsaustauschanforderungen - Bauphysik	in Arbeit	Tab. 2
VDI 3805	Produktdatenaustausch in der Technischen Gebäudeausrüstung	Reihe	Tab. 1
VDI 4645	Heizungsanlagen mit wärmepumpen in ein- und mehrfamilienhäusern - planung, errichtung, betrieb	2023-04	
VDI 6026-1	Dokumentation in der technischen Gebäudeausrüstung (TGA) - Inhalte und Beschaffenheit von Planungs-, Ausführungs- und Revisionsunterlagen	2022-08	
VDI 6070	Raumbuch	Reihe	
VDI 6070-1	Raumbuch - Allgemeine Anforderungen und Grundlagen	2023-01 Entwurf	

Verzeichnis wissenschaftlicher Quellen

- AGEB, Hrsg. 2022. „Anwendungsbilanzen zur Energiebilanz Deutschland - Endenergieverbrauch nach Energieträgern und Anwendungszwecken 2020 und 2021“. Arbeitsgemeinschaft Energiebilanzen.
- , Hrsg. 2023. „Energieflussbild 2022 für die Bundesrepublik Deutschland in Petajoule“. Arbeitsgemeinschaft Energiebilanzen.
- Ahn, Ki-Uhn, Young-Jin Kim, Cheol-Soo Park, Inhan Kim, und Keonho Lee. 2014. „BIM interface for full vs. semi-automated building energy simulation“. *Energy and Buildings*. <https://doi.org/doi.org/10.1016/j.enbuild.2013.08.063>.
- Ali, und Stanislav Kralin. 2019. „OWL Ontology: how to write a complement class definition?“ 2019. <https://stackoverflow.com/questions/54806697/owl-ontology-how-to-write-a-complement-class-definition>.
- Alwalidi, Margarita, Kheybari, und Hoffmann. 2022. „MULTI-DIMENSIONAL ASSESSMENT OF HIGH-PERFORMANCE STATIC AND DYNAMIC GLAZING FOR ENVELOPE REFURBISHMENTS“. In *BauSIM 2022*. Weimar.
- Andriamamonjy, Ando, Dirk Saelens, und Ralf Klein. 2019. „A combined scientometric and conventional literature review to grasp the entire BIM knowledge and its integration with energy simulation“. *Journal of Building Engineering*. <https://doi.org/10.1016/j.jobe.2018.12.021>.
- Andriamamonjy, Dirk Saelens, und Ralf Klein. 2018. „An automated IFC-based workflow for building energy performance simulation with Modelica“. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2018.03.019>.
- Baedlung. 2020. „Graph Data Structures“. 2020. <https://www.baeldung.com/cs/graphs>.
- Balaji, Bharathan, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, u. a. 2018. „Brick : Metadata schema for portable smart building applications“. *Applied Energy*. <https://www.sciencedirect.com/science/article/pii/S0306261918302162>.
- Bastian, Heymann, und Jacomy. 2009. „Gephi: An Open Source Software for Exploring and Manipulating Networks“. In . <http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154>.
- BauInfoConsult, Hrsg. 2022. „BIM-Monitor“. <https://www.tga-fachplaner.de/meldungen/bim-monitor-2022-23-digitalisierung-im-bau-wie-bereit-ist-deutschland>.
- Bazjanac, Vladimir, und Tobias Maile. 2004. „IFC HVAC interface to EnergyPlus - A case of expanded interoperability forenergy simulation“. In *IBPSA-USA national conference boulder*.
- Beetz, Jakob, Leon van Berlo, Ruben de Laat, und Pim van den Helm. 2010. „Bimserver.org – an open source ifc model server“. In *CIB W78 2010 cairo*.
- Benz, Alexander, Mara Geske, und Conrad Voelker. 2022. „MACHINE LEARNING FOR IMAGE-BASED RECOGNITION OF BUILDING AGE FOR URBAN ENERGY SIMULATION – TESTING AND VALIDATION ON AN EXEMPLARY CITY QUARTER“. In *BauSIM 2022*. Weimar.
- Bhattacharya, Ploennigs, und Culler. 2015. „Short paper: Analyzing metadata schemas for buildings: The good, the bad, and the ugly“. In *ACM international conference 2nd*. <https://doi.org/10.1145/2821650.2821669>.
- BMJ, Hrsg. 2013. „Verordnung über die Honorare für Architekten- und Ingenieurleistungen“. https://www.gesetze-im-internet.de/hoai_2013/.
- Borrman, König, Koch, und Beetz, Hrsg. 2014. *Building Information Modeling*.
- Bracht, M. K., A.P. Melo, und R. Lamberts. 2021. „A metamodel for building information modeling-building energy modeling integration in early design stage“. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2020.103422>.
- Braune, Hagenlocher, Quante, Ruiz Duran, und Schleife. 2022. „Wegweiser klimapositiver Gebäudebestand“. DGNB.
- Cao, Jun. 2018. „SimModel transformation middleware for modelica-based building EnergyModeling and simulation“. PhD, RWTH E3D.
- Cao, Tobias Maile, James O'Donnell, Reinhard Wimmer, und Christoph van Treeck. 2014. „Model Transformation From SIMMODEL To Modelica For Building Energy Performance Simulation“. In . https://publications.ibpsa.org/proceedings/bausim/2014/papers/bausim2014_1143.pdf.
- Challagulla, Jayakanth. 2023. „OpenSource tools to generate BIM models for application in HVAC plant simulation“. Masterarbeit, Bauhausuniversität Weimar. <https://doi.org/10.24406/publica-1433>.

- Charpenay, Victor, Sebastian Käbisch, Darko Anicic, und Harald Kosch. 2015. „An ontology design pattern for IoT device tagging systems“. In *2015 5th international conference on the internet of things (IOT)*. <https://ieeexplore.ieee.org/document/7356558>.
- CIB, TU Dresden, Hrsg. 2014. „BIM filter toolbox“. <https://openeebim.bau.tu-dresden.de/openeebim/software/BimFit-2.0-FINAL.jar>.
- Ciccozzi, de Rubeis, Paoletti, und Ambrosini. 2023. „BIM to BEM for Building Energy Analysis: A Review of Interoperability Strategies“. *energies*. <https://doi.org/10.3390/en16237845>.
- Compton, Michael, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, u. a. 2012. „The SSN ontology of the W3C semantic sensor network incubator group“. *Web Semantics*. <https://www.sciencedirect.com/science/article/pii/S1570826812000571>.
- Cozza, Stefano, Chambers, Brambilla, und Patel. 2021. „In search of optimal consumption: A review of causes and solutions to the Energy Performance Gap in residential buildings“. *Energy & Buildings*. <https://doi.org/10.1016/j.enbuild.2021.111253>.
- Cyganial, Richard. 2012. „Buildings and rooms vocabulary“. 2. Oktober 2012. <http://vocab.deri.ie/rooms>.
- Daniele, den Hartog, und Roes. 2015. „Created in close interaction with the industry: The smart appliances REference (SAREF) ontology“. In *International workshop formal ontologies meet industries*. https://doi.org/10.1007/978-3-319-21545-7_9.
- Delgoshaei, Austin, und Veronica. 2017. „Semantic models and rule-based reasoning for fault detection and diagnostics: Applications in heating, ventilating and air conditioning systems“. In *ICONS 2017: The twelfth international conference on SystemsAt: Venice, italy*. https://personales.upv.es/thinkmind/dl/conferences/icons/icons_2017/icons_2017_3_30_40028.pdf.
- Demri, Stephane, und Karin Quaas. 2021. „Concrete Domains in logics: a survey“. *ACM SIGLOG News*. <https://doi.org/10.1145/3477986.3477988>.
- . 2023. „First Steps Towards Taming Description Logics with Strings“. In *JELIA*. Dresden. https://doi.org/10.1007/978-3-031-43619-2_23.
- Di Biccari, Calcerano, D’Uffizi, Esposito, Campari, und Gigliarelli. 2023. „Building information modeling and building performance simulation interoperability: State-of-the-art and trends in current literature“. *Advanced Engineering Informatics*. <https://doi.org/10.1016/j.aei.2022.101753>.
- Dibowski, Henrik. 2013. „Semantischer gerätebeschreibungsansatz für einen automatisierten entwurf von raumautomationssystemen“. Phd, TUD; TIS.
- Eckstädt, Elisabeth. 2021. „Bidirectional coupling of building information modeling and building simulation unsing ontologies“. In *EG-ICE 2021 workshop on intelligent computing in engineering*. <http://dx.doi.org/10.14279/depositonce-12021>.
- Eckstädt, Elisabeth, und Jan Bräunig. 2024. „Bivalent Heat Pump Plant serving a Multi-Family Dwelling - Quantification of the Impact of Different Control Strategies and Parameterisations“. In *BauSIM 2024 (akzeptiert)*. Wien.
- Eckstädt, Elisabeth, Chenzi Huang, Claudia Liersch, Anne Paepcke, René Hoch, Andreas Nicolai, John Grunewald, u. a. 2023. *FMI4BIM - Standardisierte Schnittstelle für Analysemodelle von Anlagen- und Gebäudekomponenten für BIM-basierte Planung und Betrieb (Abschlussbericht)*. <https://doi.org/10.24406/publica-1997>.
- Eckstädt, Elisabeth, Karsten Menzel, Herve Pruvost, und Dirk Mayer. 2023. „Representing Modelica Models as Knowledge Graphs using the MoOnt Ontology“. In *EC3 2023*. Crete. <https://doi.org/10.24406/publica-702>.
- Eckstädt, Elisabeth, André Schneider, Anne Paepcke, Andreas Nicolai, Alexander Hentschel, und Falk Schumann. 2020. „Simulationsszenarien für Gebäudeenergiesimulation in frühen Planungsphasen“. In *BauSIM 2020*. Graz. <https://doi.org/10.3217/978-3-85125-786-1>.
- Elisabeth Eckstädt, Claudia Liersch, Alexander Hentschel, René Hoch, Danny Borchert, Milad Khalili. 2022. „COMPARING BIM2SIM WORKFLOWS FOR COUPLED BUILDING AND HVAC SIMULATION“. In *BauSIM 2022*. <https://publications.ibpsa.org/conference/?id=bausim2022>.
- Elmers, Saskia, und Alexander Hollberg. 2022. „MACHINE LEARNING FOR BUILDINGS’ ENERGY CONSUMPTION PREDICTION IN EARLY DESIGN PHASES“. In *BauSIM 2022*. Weimar.
- Esser, Sebastian, Simon Vilgertshofer, und Andre Borrmann. 2021. „Graph-based version control for asynchronous BIM level 3 collaboration“. In *EG-ICE 2021*.
- Esser, Sebastian, Simon Vilgertshofer, und André Borrmann. 2022. „Graph-based version control for asynchronous BIM collaboration“. *Advanced Engineering Informatics*. <https://doi.org/10.1016/j.aei.2022.101664>.

- . 2023. „Version control for asynchronous BIM collaboration: Model merging through graph analysis and transformation“. <https://doi.org/10.1016/j.autcon.2023.105063>.
- Feeney, Kevin. 2019. „<https://medium.com/terminusdb/graph-fundamentals-part-3-graph-schema-languages-1fc25ca294dd>“. 2019. <https://medium.com/terminusdb/graph-fundamentals-part-3-graph-schema-languages-1fc25ca294dd>.
- Frahm, Moritz, Elena Klumpp, Meisenbacher, Matthes, Mikut, und Hagenmeyer. 2022. „Development and Validation of Grey-Box Multi-Zone Thermal Building Models“. In *BauSIM 2022*. Weimar.
- Fritzson, Peter, Adrian Pop, Abdelhak, Adeel Ashgar, Bachmann, Willi Braun, Bouskela, und Robert Braun. 2020. „The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development“. *Modeling, Identification and Control* 41 (4). <https://www.mic-journal.no/ABS/MIC-2020-4-1.asp/>.
- Fruchterman, und Reingold. 1991. „Graph drawing by force-directed placement“. *Software: Practice and Experience* 21 (11). <https://doi.org/10.1002/spe.4380211102>.
- Gao, Hao, Christian Koch, und Yupeng Wu. 2019. „Building information modelling based building energy modelling: A review“. *Applied Energy*. <https://doi.org/10.1016/j.apenergy.2019.01.032>.
- Geiger, Nichersu, Häfele, Hagenmeyer, und Koenigsdorff. 2022. „USAGE PROFILE ENRICHMENT OF CITYGML MODELS FOR URBAN BUILDING ENERGY MODELING“. In *BauSIM 2022*. Weimar.
- Gerold, Fabian. 2014. „IfcPlusPlus“. <http://www.ifcquery.com/>.
- Giannakis. 2015. „A methodology to automatically generate geometry inputs for energy performance simulation from IFC BIM models“. In *IBPSA 2015 hyderabad*. University of Crete; Fraunhofer IBP.
- . 2019. „A workflow and for automated and building energy and performance model and generation“. In . University of Crete; UC London.
- Glimm, Horrocks, Motik, Stoilos, und Wang. 2014. „HermiT: An OWL 2 Reasoner“. *Journal of Automated Reasoning*. <https://link.springer.com/article/10.1007/s10817-014-9305-1>.
- „Graphisoft Building Systems GmbH, ,OPEN BIM und IFC,’ Data Design System GmbH“ o. J. Zugriffen 9. August 2019. <https://www.dds-cad.de/produkte/ihr-mehrwert/open-bim-und-ifc/>.
- Haller, Armin, Krzysztof Janowicz, Simon J D Cox, Maxime Lefrançois, Kerry Taylor, Danh Le Phuoc, Joshua Lieberman, Raúl García-Castro, Rob Atkinson, und Claus Stadle. 2018. „The SOSA/SSN ontology: A joint W3C and OGC standard specifying the semantics of Sensors, observations, actuation, and Sampling“. In *Semantic web X (2018)* 1.
- Herrmann, RAffael. 2012. *Wissenspyramide*. www.derwirtschaftsinformatiker.de.
- Hirsch, Hauke, und Anne Paepcke. 2022. „DETAILED MODELLING OF LARGE DISTRICT HEATING AND COOLING NETWORK COUPLED TO GROUND HEAT EXCHANGER“. In *BauSIM 2022*. Weimar.
- Hu, Yifan. 2005. „Efficient and high quality force-directed graph drawing“. *The Mathematica Journal*.
- Huang, Chenzi, und Elisabeth Eckstädt. 2023. „Comparison of different coupling variants for building and HVAC simulation“. In *CISBAT 2023*. Lausanne. <https://doi.org/10.1088/1742-6596/2600/7/072005>.
- „IBPSA Project 1“. 2021. 2021. <https://ibpsa.github.io/project1/index.html>.
- Imam, Salah, David A Coley, und Ian Walker. 2017. „The building performance gap: Are modellers literate?“ *Building Services Engineering Research & Technology*. <https://doi.org/10.1177/0143624416684641>.
- Ismail, Ali. 2011. „IfcWebServer“. <https://ifcwebserver.org/>.
- Jacomy, Venturini, Heymann, und Bastian. 2014. „ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software“. *PLoS one* 9 (6). <https://doi.org/10.1371/journal.pone.0098679>.
- Jeong, WoonSeong, Jong Bum Kim, Mark J. Clayton, Jeff S. Haberl, und Wei Yan. 2014. „Translating building information modeling to building energy modeling using model view definition“. *The Scientific World Journal*. <https://doi.org/10.1155/2014/638276>.
- Jorissen, Filip, Glenn Reynders, Ruben Baetens, Damien Picard, Dirk Saelens, und Lieve Helsen. 2018. „Implementation and verification of the IDEAS building energy simulation library“. *Journal of Building Performance Simulation* 11 (6): 669–88. <https://doi.org/10.1080/19401493.2018.1428361>.
- Kaiser, Jens, und Pit Stenzel. 2015. „eeEmbedded D4.2: Energy system information model - ESIM“.
- Kamada, und Kawai. 1989. „An algorithm for drawing general undirected graphs“. *Information Processing Letters* 31 (1). [https://doi.org/10.1016/0020-0190\(89\)90102-6](https://doi.org/10.1016/0020-0190(89)90102-6).
- Kamel, Ehsan, und Ali Memari. 2018. „Review of BIM’s application in energy simulation: Tools, issues, and solutions“. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2018.11.008>.

- Katranuschkov, Peter, Florian Heinzig, Jens Kaiser, Byron Protopsaltis, Mathias Kadolsky, Ken Baumgärtel, Robert Schülbe, u. a. 2016. „eeEmbedded – D6.2 multimodel mapper – simulation model generator“.
- Krijnen, Thomas. 2011. „IfcOpenShell“. <http://ifcopenshell.org/> <https://github.com/IfcOpenShell>.
- Kukkonen. 2022. „An ontology to support flow system descriptions from design to operation of buildings“. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2021.104067>.
- Lamy, Jean-Baptiste. 2017. „Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies“. *Artificial Intelligence In Medicine*.
- Lee, Edward A., und S. Neuendorffer. 2000. „MOML-A modeling markup language in XML-Version 0.4“. EECS Department, University of California, Berkeley. <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2000/3818.html>.
- Leimbach, Nils. 2022. „Semantikbasierte Entwurfsautomatisierung gebäudetechnischer Anlagen“. Studienarbeit, TU Dresden. <https://doi.org/10.24406/publica-265>.
- „Level of development (LOD) specification“. 2013.
- Li, Hang, und Jiansong Zhang. 2023. „Improving IFC-Based Interoperability between BIM and BEM Using Invariant Signatures of HVAC Objects“. *Journal of Computing in Civil Engineering*. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0001063](https://doi.org/10.1061/(ASCE)CP.1943-5487.0001063).
- Li, und Jiansong Zhang. 2021. „Interoperability between BIM and BEM using IFC“. In . Reston, VA, USA. <https://doi.org/10.1061/9780784483893.078>.
- Li, Jiansong Zhang, Xue, Debs, Chang, Qu, Sparkling, und Goldwasser. 2022. „Issues in Bi-Directional Interoperability between BIM and BEM“. In . Arlington, VA, USA. <https://doi.org/10.1061/9780784483961.142>.
- Liu, Han, Gao, und Gu. 2023. „A Parallel IFC Normalization Algorithm for Incremental Storage and Version Control“. In *EG-ICE 2023*. <https://doi.org/10.48550/arXiv.2312.14931>.
- Lockley, Steve. 2007. „xBIM toolkit“. <https://docs.xbim.net/>.
- „LOD Australian NATSPEC National BIM Guide“. 2011.
- Lutz, und Brandt. 2007. „A Tableau Algorithm for Description Logics with Concrete Domains and General TBoxes“. *Journal of Automated Reasoning*. <https://doi.org/10.1007/s10817-006-9049-7>.
- Maier, Laura, David Jansen, Fabian Wüllhorst, Martin Kremer, Alexander Kümpel, Tobias Blacha, und Dirk Müller. 2023. „AixLib: an open-source Modelica library for compound building energy systems from component to district level with automated quality management“. *Journal of Building Performance Simulation*. <https://doi.org/10.1080/19401493.2023.2250521>.
- Manotas, Miguel. 2021. „Untersuchungen zur Simulation raumluftechnischer Anlagen zur Unterstützung von Planung und Betrieb von Nichtwohngebäuden“. Studienarbeit, TU Dresden. <https://doi.org/10.24406/EAS-N-634732>.
- Martin, Brown, Klavans, und Boyack. 2011. „OpenOrd: An Open-Source Toolbox For Large Graph Layout“. In . <https://doi.org/10.1117/12.871402>.
- Mitterhofer, Matthias. 2018. „A methodology for a scalable building performance simulation based on modular components“. Dissertation, Fraunhofer IBP; TU München. http://publica.fraunhofer.de/eprints/urn_nbn_de_0011-n-4904932.pdf.
- Musen. 2015. „The Protégé project: A look back and a look forward“. *AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence*. <https://doi.org/10.1145/2557001.25757003>.
- Nachawati, Mohamad Omar, Gianmaria Bullegas, Andrey Vasilyev, Joe Gregory, Adrian Pop, Maged Elasar, und Adeel Ashgar. 2022. „Towards an open platform for democratized model-BasedDesign and engineering of cyber-physical systems“. In *American modelica conference*. <https://doi.org/10.3384/ECP21186>.
- Nasyrov, Stratbücker, Ritter, Borrmann, Hua, und Lindauer. 2014. „Building information models as input for building energy performance simulation - the current state of industrial implementations“. In *ECPMM 2014*. <https://doi.org/10.1201/b17396-80>.
- Nasyrov, Vladislav. 2013. „Building Information Models als Input für Energetische Gebäudesimulation“. Masterarbeit, TUM. https://publications.cms.bgu.tum.de/theses/nasyrov_2013_ritter.pdf.
- Noll, Josef. 2013. „Comparison of Pellets, FaCT and HermiT“. 2013. https://its-wiki.no/wiki/Comparison_of_Pellets,_FaCT_and_HermiT.
- Norrefeldt, Victor. 2013. „VEPZO – Velocity Propagating Zonal Model – A locally refined airflow model for confined spaces to use in optimization applications“. Phd, Fraunhofer IPB; Universität Stuttgart.
- Novak und Sindelar. 2011. „Applications of ontologies for assembling simulation models of industrial systems“. In *OTM confederated international conferences „On the Move to Meaningful Internet Systems“*. https://publik.tuwien.ac.at/files/PubDat_202236.pdf.

- Nytsch-Geusen. 2019. „BIM2Modelica-An open source toolchain for generating and simulationg thermal multi-zone building models by using structured data from BIM models“. In *International Modelica Conference 13th , Regensburg, Germany*. <https://modelica-buildingsystems.de/pub/ModelicaConference2019.pdf>.
- Nytsch-Geusen, Christoph, Jörg Huber, Manuel Ljubijankic, und Jörg Rädler. 2012. „Modelica BuildingSystems - Eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme“. In .
- Odeh, Khaleel, und Pieter de Wilde. 2020. „BIM to BEM: an investigation of practical interoperability challenges when working with revit and DesignBuilder“. In *EG-ICE 2020*. <https://doi.org/10.14279/depositonce-9977>.
- O'Donnell, J., Richard See, C. Rose, Maile, V. Bazjanac, und P. Haves. 2011. „SimModel: A domain data model for whole building energy simulation“. In *IBPSA 2011*.
- O'Sullivan, Barry, und Marcus Keane. 2005. „Specification of an ifc based intelligent graphical user interface to support building energy simulation“. *IBPSA 2005*.
- „OWL 2 Referenz“. 2012. <https://www.w3.org/2012/pdf/REC-owl2-profiles-20121211.pdf>.
- „OWL Implementations“. 2020. 2020. <https://www.w3.org/2001/sw/wiki/OWL/Implementations>.
- „OWL Referenz“. 2004. <https://www.w3.org/TR/owl-ref/>.
- Paepcke, Anne, Martin Leuschke, und René Hoch. 2022. „STABILE SIMULATION THERMOHYDRAULISCHER VERBRAUCHERNETZWERKE BEI EXTERNER PUMPENREGELUNG“. In *BauSIM 2022*. Weimar.
- Parr, Terence. 2013. *The Definitive Antlr 4 Reference*. <https://pragprog.com/titles/tpantlr2/the-definitive-antlr-4-reference/>.
- Pauen, Nicolas. 2022. „Graphbasierte Algorithmen und gesamtheitliche Repräsentation von Systemen der TGA mit BIM und Linked Data“. Phd, RWTH.
- Pauen, Nicolas, Ville Kukkonen, Ali Küçükavci, Mads Holten Rasmussen, Mikki Seidenschnur, Dominik Schlüter, Christian Anker Hviid, und Christoph van Treeck. 2022. „A roadmap toward a unified ontology for buildingservice systems in the AECO industry: TSO and FSO“. In *LDAC 2022*. <https://ceur-ws.org/Vol-3213/paper05.pdf>.
- Pauwels, Pieter. 2020. „IFCtoRDF“. <https://github.com/pipauwel/IFCtoRDF>.
- Pauwels und Roxin. 2016. „SimpleBIM: From full ifcOWL graphs to simplified building graphs“. In *ECPPM 2016*. <https://biblio.ugent.be/publication/8041826/file/8070159.pdf>.
- Pauwels und Terkaj. 2016. „EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology“. *Automation in Construction*.
- Pauwels, Chi Zhang, und Lee. 2017. „Semantic web technologies in AEC industry: A literature overview“. *Automation in Construction*. <https://www.sciencedirect.com/science/article/abs/pii/S0926580516302928>.
- Pawar, Priya, Boranian, und Lang. 2022. „SPACE, THERMAL COMFORT AND ENERGY PERFORMANCE OF A MIXED-MODE OFFICE BUILDING IN A HOT AND HUMID CLIMATE“. In *BauSIM 2022*. Weimar.
- Pinheiro, Sergio, Reinhard Wimmer, James O'Donnell, Sergej Muhic, Vladimir Bazjanac, Tobias Maile, Jérôme Frisch, und Christoph van Treeck. 2018. „MVD based information exchange between BIM and building energy performance simulation“. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2018.02.009>.
- Pinto, Helena Sofia, und João P. Martins. 2004. „Ontologies: How can they be built?“ *Knowledge and Information Systems*.
- Ploennigs, Burkhard Hensel, Henrik Dibowski, und Klaus Kabitzsch. 2012. „BASont - A modular, adaptive building automation system ontology“. In *IECON 2012*.
- Pop, Adrian, und Peter Fritzson. 2004. „The modelica standard library as an ontology for modeling and simulation of physical systems“. <https://www.ida.liu.se/~adrpo33/reports/adrpo-petfr-WS-OSEA.pdf>.
- Pop und Fritzson. 2003. „ModelicaXML: A modelica XML representation with applications“. In *International modelica conference 3rd*. <https://www.ida.liu.se/~petfr27/paperlinks/2003-11-pop-fritzson-modelica2003-modelicaxml.pdf>.
- Porsani, del Valle de Lersundi, Sachez-Osteiz Gutierrez, und Bandera. 2021. „Interoperability between Building Information Modelling (BIM) and Building Energy Model (BEM)“. *Applied Sciences*. <https://doi.org/10.3390/app11052167>.
- Pruvost, Herve. 2022. „SENSE - A semantic composition that conceptualizes building energy systems in their operation phase“. 2022. w3id.org/sense.
- Rasmussen, Frausing, Hviid, und Karlshoj. 2018. „Demo: Integrating building information modeling and sensor observations using semantic web“. In *Semantic sensor networks workshop 2018At: Monterey, CA, united states*. https://orbit.dtu.dk/files/154895068/Untitled_2.pdf.

- Rasmussen, PAuwels, Lefrancois, Schneider, Hvidd, und Karlshoj. 2017. „Recent changes in the building topology ontology“.
- Reinisch, Kofler, und Kastner. 2010. „ThinkHome: A smart home as digital ecosystem“. In *IEEE DEST 2010*. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5610636>.
- Remmen, Jun Cao, Ebertshäuser, Jérôme Frisch, Moritz Lauster, Tobias Maile, James O'Donnell, u. a. 2015. „An Open Framework For Integrated BIM-Based Building Performance Simulation Using Modelica“. In *IBPSA 2015 hyderabad*. <https://doi.org/10.26868/25222708.2015.2384>.
- Roxin, Ana, Vishak Dundee, und Vladimir Vukovic. 2021. „Investigating potential alignments between modelica standard library and SAREF ontologies“. In *LDAC2021*.
- Scherer, Raimar J., und Sven-Eric Schapke, Hrsg. 2014. *Informationssysteme im Bauwesen 1*. Springer.
- Schneider, Georg Ferdinand. 2017. „Towards aligning domain ontologies with the Building Topology Ontology“. In *LDAC2017*. http://www.linkedbuildingdata.net/ldac2017/files/Presentations/LDAC2017_georgschneider.pdf.
- . 2019a. „Automated ontology matching in the architecture, engineering and construction domain - a case study“. In *LDAC2019-7th linked data in architecture and construction workshop lisboa*. <https://ceur-ws.org/Vol-2389/03paper.pdf>.
- . 2019b. „Semantic modelling of control logic in automation systems“. Phd, KIT. <https://publikationen.bibliothek.kit.edu/1000092405/24447773>.
- Schneider, Kontes, Qiu, Silva, Bucur, Malanik, Schindler, u. a. 2020. „Design of knowledge-based systems for automated deployment of building management services“. *Automation In Construction*.
- Schneider, Pauwels, und Steiger. 2017. „Ontology-based modeling of control logic inBuilding automation systems“. *IEEE Transactions on Industrial Informatics*. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8015195>.
- Shani, Raiend, Wadler, Hedberg, Wagner, D'Angelo, und Tronarp. 2014. „SPRINT software platform for integration of engineering and things D5.11 final report“. http://www.sprint-iot.eu/public_deliverables/D_5_11.PU.Final_Report.pdf.
- Shani, Uri. 2017. „Can ontologies prevent MBSE models from becoming obsolete?“ Herausgegeben von IEEE. <https://core.ac.uk/download/pdf/144853941.pdf>
<https://zenodo.org/record/582774/files/obsolescence.syscon.17.1570318990.shani.final.pdf>.
- Silver, Hassan, und Miller. 2007. „From domain ontologies to modeling ontologies to executable simulation models“. In *Proc. Winter simulation conference 2007*.
- Sindelar, Radek, und Petr Novak. 2016. „Ontology-Based Simulation Design and Integration“. In *Semantic Web Technologies for Intelligent Engineering Applications*. Springer Cham. <https://link.springer.com/book/10.1007/978-3-319-41490-4>.
- Stratbücker, Sebastian, Matthias Mitterhofer, Gesa Benndorf, Marcel Dang, und Nicolas Rehault. 2017. „EnEff-BIM: Planung, Auslegung und Betriebsoptimierung von energieeffizienten Neu- und Bestandsbauten durch Modellierung und Simulation auf Basis von Bauwerkinformationsmodellen : Abschlussbericht 2“. Fraunhofer IPB; Fraunhofer ISE. <https://doi.org/10.2314/GBV:895388456>.
- Strnadl, Peter Katranuschkov, Karsten Menzel, Mischa Sethi, und Beatrix Weber. 2024. „A Gaia-X-Based Decentralized Platform Architecture for a Distributed Digital Twin in the AECO Sector“. *Journal of Information Technology in Construction (submitted)*.
- Strnadl, und Schöning. 2023. „Datenplattformen, Datenräume und (Daten-)Ökosysteme – (IT) Architekturen und Praxis des Datenteilens“. In .
- Tauscher, Helga. 2016. „Configurable nD-visualizationfor complex BuildingInformation models“. Phd, TUD; CIB. <https://tud.qucosa.de/api/qucosa:30539/attachment/ATT-0/>.
- Teclaw, Oraskari, Rasmussen, Labonne, und Hjelseth. 2023. „Building system data integration using semantic“. In . https://www.ucl.ac.uk/bartlett/construction/sites/bartlett_construction/files/building_system_data_integratation_using_semantic.pdf.
- Terkaj, Schneider, und Pauwels. 2017. „Reusing domain ontologies in LinkedBuilding data: the case of BuildingAutomation and control“. In *Workshop formal ontologies meet industry, joint ontology workshops 2017, CEUR workshop proceedings*.
- Tomasevic, Batic, Blanes, Keane, und Vranes. 2015. „Ontology-based facility data model for energy management“. *Advanced Engineering Informatics*. <https://www.sciencedirect.com/science/article/abs/pii/S1474034615000981>.

- Treecck, Christoph van, Elixman, Rudat, Hiller, Herkel, und Berger. 2016. *Gebäude. Technik. Digital - building information modeling.* Herausgegeben von Viega. Springer Vieweg Berlin, Heidelberg. <https://link.springer.com/book/10.1007/978-3-662-52825-9>.
- Treecck, Christoph van, Dirk Müller, Petra von Both, Christoph Nytisch-Geusen, Tobias Maile, und Reinhard Wimmer. 2017. „EnEff-BIM: Planung, Auslegung und Betriebsoptimierung von energieeffizienten Neu- und Bestandsbauten durch Modellierung und Simulation auf Basis von Bauwerkinformationsmodellen : Abschlussbericht“. RWTH E3D; RWTH EBC; KIT; UdK; AEC3. <https://doi.org/10.2314/GBV:89124431X>.
- Urbanski, Matthias. 2021. „Untersuchungen zur Kopplung von BIM mit semantischen Technologien am Beispiel einer Kälteerzeugungsanlage“. Masterarbeit, TH Köln. 10.24406/eas-n-640382.
- van Dronkelaar, Dowson, Burman, Spataru, und Mumovic. 2016. „A Review of the energy Performance Gap and its Underlying Causes in Non-Domestic Buildings“. *frontiers in Mechanical Engineering*. <https://doi.org/10.3389/fmech.2015.00017>.
- Villalón, María Poveda, und Raúl García Castro. 2017. „SmartM2M;Smart appliances extension to SAREF; part 3: Building DomainETSITS 103_410-3 V1.1.1 (2017-01)“. Universidad Politécnica de Madrid. https://www.etsi.org/deliver/etsi_ts/103400_103499/10341003/01.01.01_60/ts_10341003v010101p.pdf <https://mariapoveda.github.io/saref-ext/OnToology/SAREF4BLD/ontology/saref4bldg.ttl/documentation/index-en.html>.
- Volkmer, Löhr, Ryba, Sedlak, Gerber, und Koenigsdorff. 2022. „A SCALABLE APPROACH TO LOAD PROFILE DETERMINATION ON A CITY DISTRICT LEVEL, APPLICABLE IN EARLY PLANNING STAGES“. In *BauSIM 2022*. Weimar.
- W3C. 2007. „Why only object properties?“ 2007. https://www.w3.org/2007/OWL/wiki/Property_Chains.
- Weiß, Dirk, Tribulowski, und Hirth. 2022. „WIRTSCHAFTLICHES KONDITIONIEREN VON GEBÄUDEN MIT REGENERATIVEN ENERGIEN – NOTWENDIGKEIT UND MEHRWERT DER GEBÄUDESIMULATION“. In *BauSIM 2022*. Weimar.
- Wetter, Michael, Milica Grahovac, und Jianjun Hu. 2018. „Control Description Language“. In . <https://simulationresearch.lbl.gov/wetter/download/2018-americanModelica-WetterGrahovacHu.pdf>.
- Wetter, Michael, und Christoph van Treecck. 2017. „New generation computational tools for building & community energy systems“. IEA EBC. <http://www.iea-annex60.org/downloads/iea-ebc-annex60-final-report.pdf>.
- Wetter, Michael, W. Zuo, Thierry Nouidui, und Xiufeng Pang. 2014. „Modelica {Buildings} library“. *Journal of Building Performance Simulation*. <https://doi.org/10.1080/19401493.2013.765506>.
- Wimmer, R., T. Maile, J. O'Donnell, J. Cao, und C vanTreecck. 2014. „Data-requirements specification to support BIM-based HVAC-definitions in Modelica“. In *BauSIM 2014*.
- Wimmer, Reinhard. 2020. „BIM-Informationsmanagement bei der Thermisch-Energetischen Simulation von gebäudetechnischen Anlagen“. PhD, RWTH E3D.
- Wolfram Research, Inc. 2014. „Wolfram modelica ontology“. 2014. <http://www.sprint-iot.eu/Wolfram-Modelica-ontology.zip>.
- Zeb, Akhtar, und Juha Kortelainen. 2021. „Web Ontology Language data modelling of Modelica simulation models“. VTT Technical Research Centre of Finland.
- Zhang, Benzcly. 2019. „IfcSTEP-to-IfcOWL converters“. 2019. <https://github.com/BenzclyZhang/IfcSTEP-to-IfcOWL-converters>.

Anhang

Digitaler Anhang

Die Verweise auf die zur Arbeit gehörigen Quelltexte und Beispieldateien wurden in den Kapiteln als Fußnoten vermerkt und werden in der folgenden Tabellen thematisch gruppiert zusammengestellt.

Tab. 57 Zusammenstellung der zur Arbeit gehörigen Quelltexte, Beispieldateien und Binaries

Bezeichnung / Beschreibung	Pfad
Ontologie MoOnt	https://github.com/ElisEck/MO-x-IFC/tree/main/TBox/ontologies/7_MoOnt
Alignment AlMAix	https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/alignments/AlMAix_0.5.0.ttl
MBL Ontologie	https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/ontologies/8_ModelicaLibraries/MBL_8.0.0-0.8.0.ttl
AixLib Ontologie	https://github.com/ElisEck/MO-x-IFC/blob/main/TBox/ontologies/8_ModelicaLibraries/AixLib_1.0.0-0.8.0.ttl
Semantic Translator	https://github.com/ElisEck/MO-x-IFC/blob/main/04_SemTran/04_SemanticTranslator.py
Procedural Translator	https://github.com/ElisEck/MO-x-IFC/blob/main/08_ProTran/08_ProceduralTranslator.py PythonLib/TranslationHelper.py
MoTTL-Transcriptor executable	https://github.com/ElisEck/MO-x-IFC/releases/ https://gitlab.cc-asp.fraunhofer.de/iis-eas-acs/ifcscripting/-/blob/main/_Projekte/EAS_KLT/IFCs/
Zwischenstände der Aufbereitung des IFC-Files für UseCase 1 BIM2SIM	20 System korrigiert/all.ifc* 20 System korrigiert/all_gerundeteKoordinaten.ifc 20 System korrigiert/all_gerundeteKoordinaten_optimized.ifc 25 Neustart ab 23b ohne unwichtige Systeme/25c.ifc 25 Neustart ab 23b ohne unwichtige Systeme/25y2.ifcs
Experiment UseCase 1	https://github.com/ElisEck/MO-x-IFC/blob/main/examples/BIM2SIM/A_Server/ debug/Server_classes_mo.ttl debug/Server_F_3D_2D.ifc Server_F.ifc Server_K.ifc Server_mo.ttl
Experiment UseCase 2	https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2BIM/D_HIL/ hil_F.ifc hil_F_3D_2D.ifc hil_ifc.ttl hil_mo.ttl
TTL2MO	https://github.com/ElisEck/MO-x-IFC/blob/main/05_TTL2MO/05_TTL2MO.py
Weiteres Beispiel aus (Eckstädt, Menzel, u. a. 2023)	https://github.com/ElisEck/MO-x-IFC/blob/main/examples/SIM2GA/C_HeatPumpPlant/LBDCG_example/HeatPumpPlant.mo https://github.com/ElisEck/MO-x-IFC/blob/public-main/src/test/java/output/ex_20221215_1154_fullclean.ttl
Hilfsfunktionen zur Aufbereitung der IFC-Files	https://github.com/ElisEck/MO-x-IFC/blob/main/PythonLib/IFC/IfcNetwork.py

* siehe auch Projektverzeichnis: i:\projekte\energiebda_180726\work\eas\1_Planungsdaten\420 430 434 Heizung Lüftung Kälte\IFC\KLT\2023-02-08 Aufbereitung Elisabeth\09-22 Gesamtgebäude\20 System korrigiert\

Handling von Wissensgraphen - Größenbegrenzungen

Tab. 58 Handling großer Textdateien – Grenzen der Werkzeuge

Werkzeug	Begrenzung für Anzeige	Begrenzung für Suche
Notepad++	850MB ttl geht nicht auf	
Windows 11 Editor	850 MB ttl geht auf	(Suche langsam)
TotalCommander Lister	850 MB ttl geht auf	(Suche zügig, aber ihr Funktionsumfang ist eingeschränkt)

Abbildungen Vereinfachung Graph im IFC

Im Folgenden sind die Schritte für die einzelnen Systeme dargestellt:

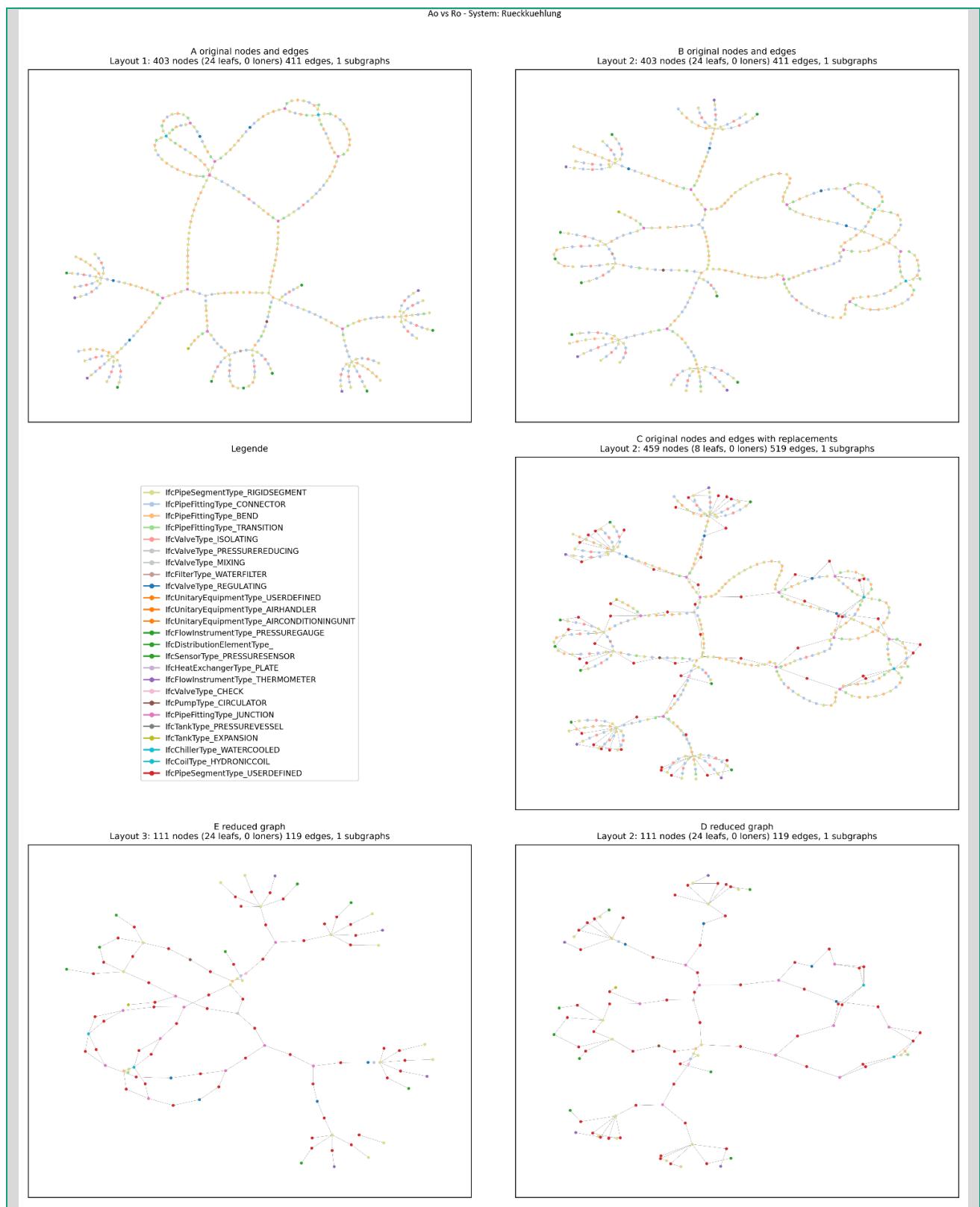


Abb. 72 Vereinfachung für System RK

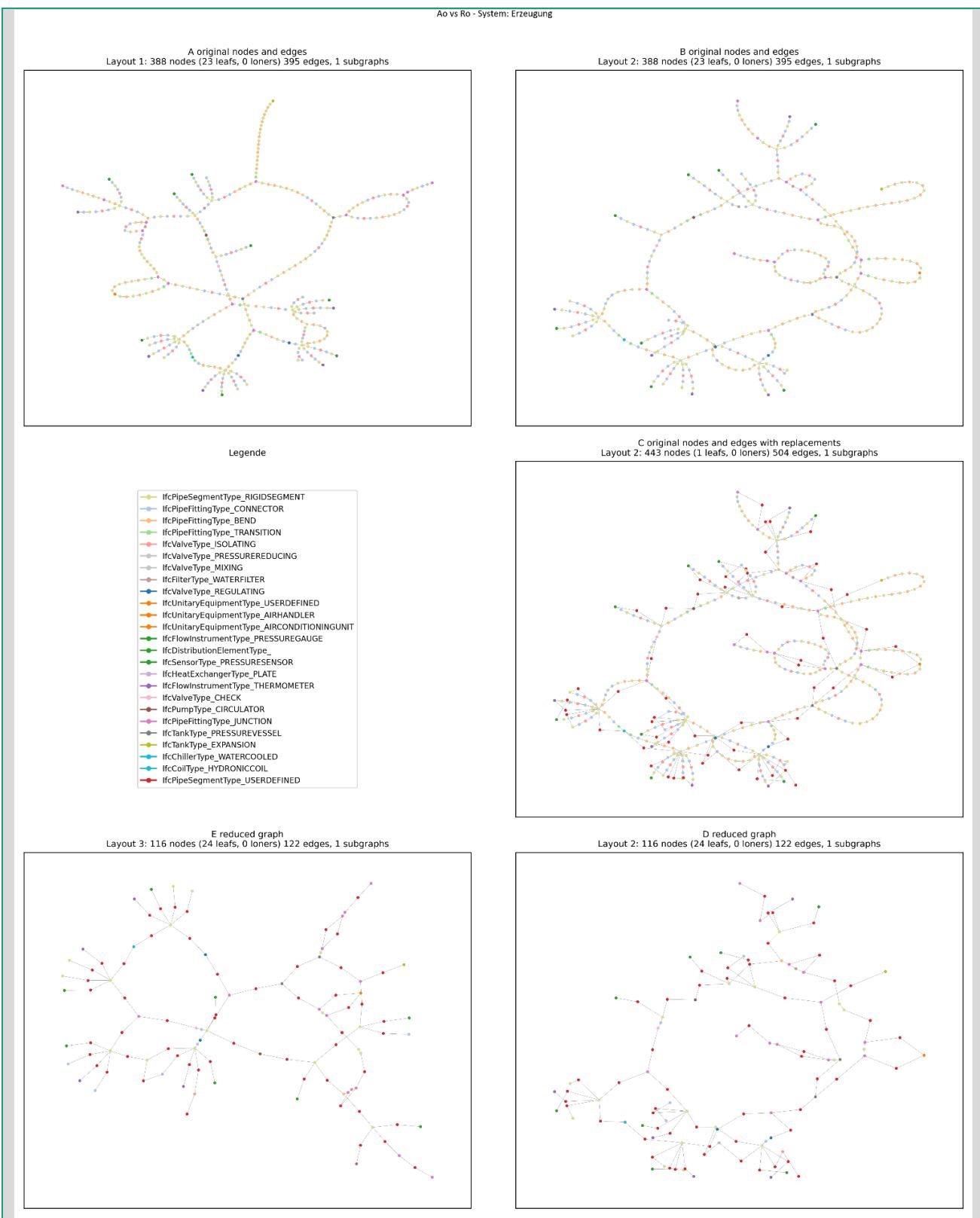


Abb. 73 Vereinfachung für System Erzeugung

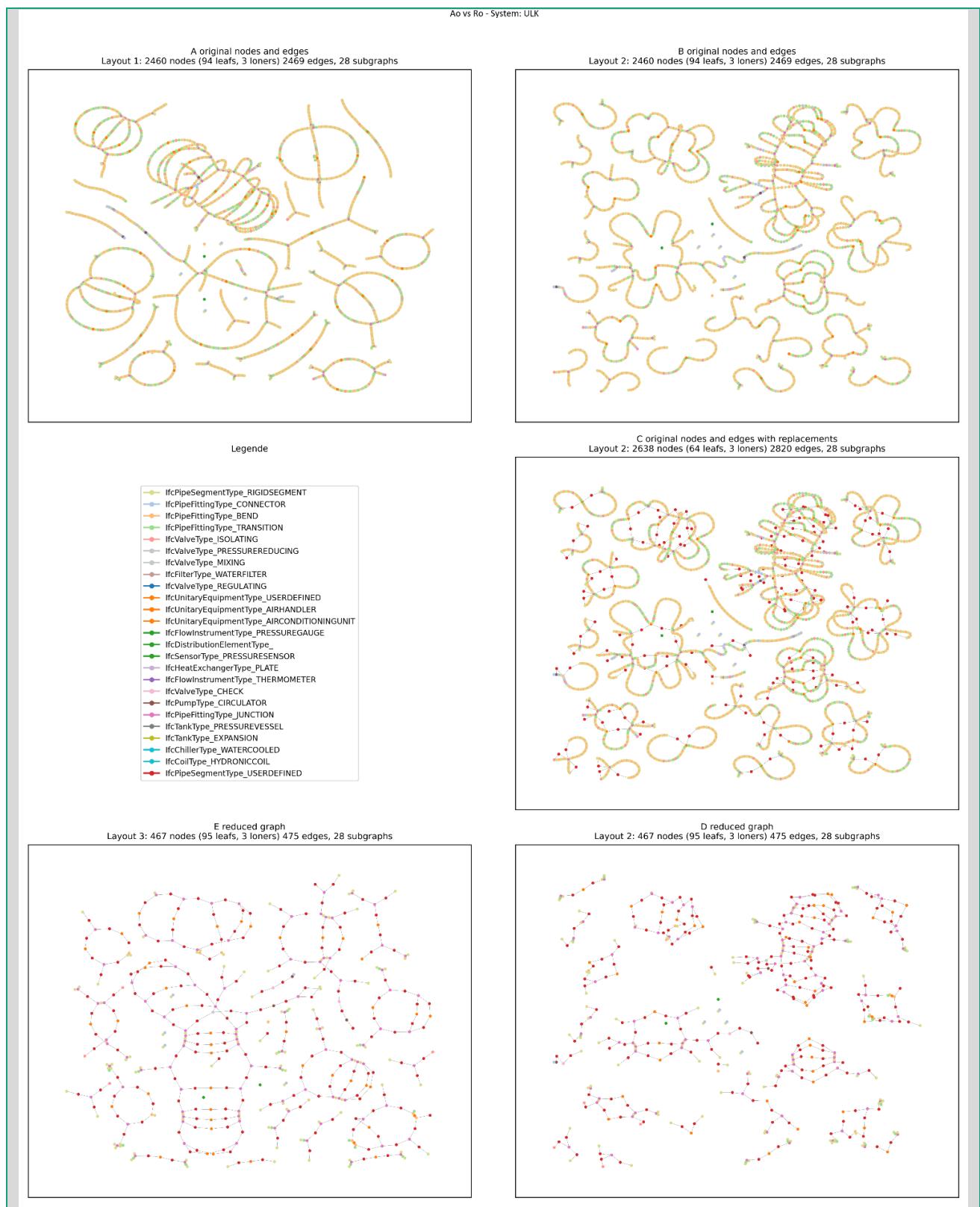


Abb. 74 Vereinfachung für System ULK

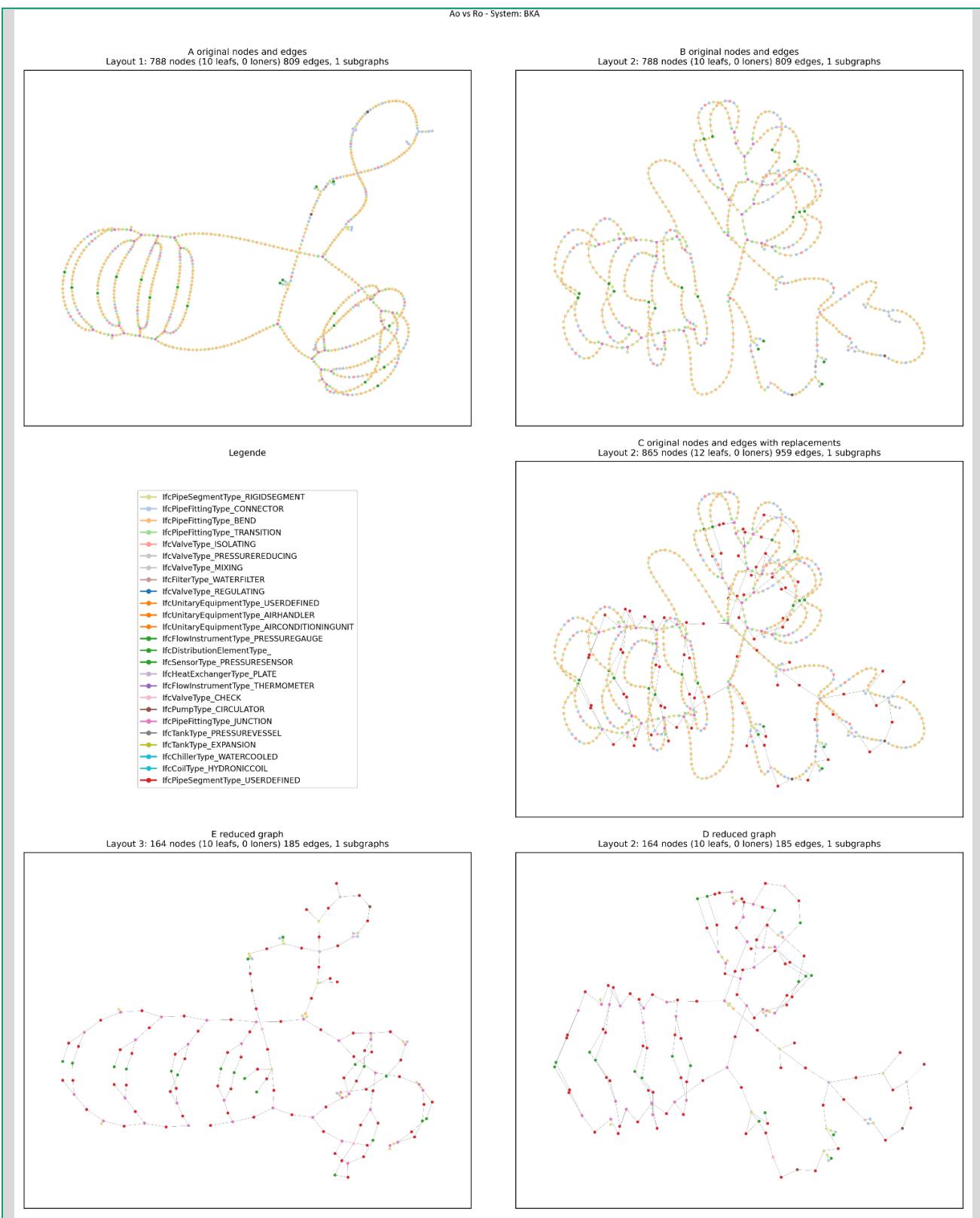


Abb. 75 Vereinfachung für System BKA

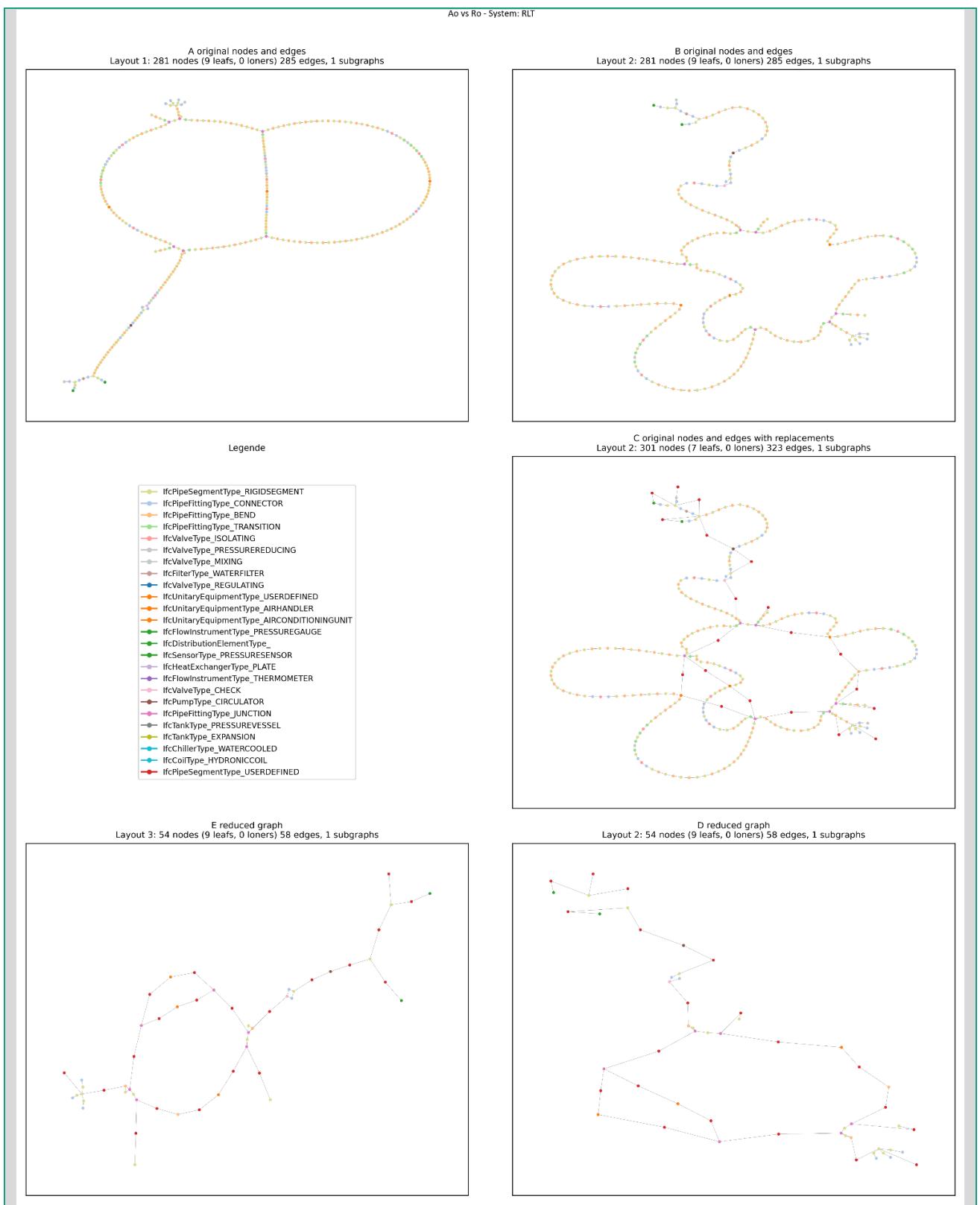


Abb. 76 Vereinfachung für System RLT