Ingenieurfakultät Bau Geo Umwelt
Fakultät für Architektur

Lehrstuhl für Energieeffizientes und Nachhaltiges Planen und Bauen
Prof. Dr.-Ing. Werner Lang

Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

Entwicklung einer Schnittstelle zwischen IFC-Gebäudemodellen und Modelica

Shan Hua

Masterthesis

für den Abschluss: Master of Science
im Studiengang Energieeffizientes und Nachhaltiges Bauen

Autor:	Shan Hua
Matrikelnummer:	[redacted]
Prüfer:	Prof. Dr.-Ing. Werner Lang
Betreuer:	M.Sc. Fabian Ritter
	Dipl.-Math. Manuel Lindauer
Ausgabedatum:	01. Okotober 2013
Abgabedatum:	13. März 2014

Abstract

Dynamic simulation provides the possibility to predict a building's thermal-energetic and ecological performance in consideration of its entire life cycle in design processes, so that the decision-makings for design, construction and project management can be supported. A number of software such as TRNSYS and EnergiePlus have been developed to perform dynamic simulations of buildings. But the manual data input of buildings and their facilities is time- and resource-consuming in most simulation software programs. The efficiency of the cooperation between involved energy engineers and architects can be improved if BIM is used in simulations with automatic data exchange.

The goal of this article is to build an interface between BIM and energy-simulation. For this purpose, the IFC building data model and the simulation language Modelica are used as objects for exchange. Thus, building data required by simulations can be called in the IFC data model and are available in Modelica-based simulation methods. In the first step, a part of the IFC data model is established as a so-called MVD after the data requirements of dynamic simulation processes. Through the established interface, the IFC part is translated into Modelica. For this purpose, this master thesis explains the methodology and procedures, and introduces a self-developed library for data retrieval in programming.

Kurzfassung

Dynamische Simulationen bieten eine Möglichkeit, die thermisch-energetischen und ökologischen Verfahren eines Gebäudes im Betracht von seinem ganzen Lebenszyklus in Planungsphasen vorherzusagen, damit die Entscheidungsprozesse der Planung, Ausführung und Projektmanagement unterstützt werden können. Vielfältige Software wie TRNSYS und EnergiePlus wurden entwickelt, um dynamische Simulationen von Gebäuden auszuführen. Aber die manuelle Datenaufnahme der Informationen von Gebäuden und technischen Anlagen ist in die meisten der Simulationssoftware zeit- und ressourcenaufwändig. Die Effizienz der Zusammenarbeit der beteiligten Energietechniker und Architekten wird erhöht, wenn Building Information Modeling (BIM) in Simulationen eingesetzt und der Datenaustausch dazwischen automatisiert werden kann.

Das Ziel der Arbeit besteht darin, eine Schnittstelle zwischen BIM und Energiesimulationen zu erstellen. Dafür wird das IFC-Gebäudedatenmodell und Simulationsprogrammsprache Modelica als auszutauschende Objekte ausgewählt. Damit können die von Simulationen erforderte Gebäudedaten in IFC-Dateien abgerufen und in Modelica-basierte Simulationsverfahren zur Verfügung gestellt werden. Ein Ausschnitt von der IFC-Datenmodell wird im ersten Schritt nach dem Datenbedarf der dynamischen Simulationsverfahren als sogenannte Model View Definition (MVD) erstellt. Durch die entwickelte Schnittstelle wird der IFC-Ausschnitt in Modelica übersetzt. Hierzu wird die Methodik, Vorgehensweise und eine selbst-entwickelte Bibliothek für den Datenabruf bei Programmierungen in dieser Masterarbeit erläutert.

Inhaltsverzeichnis

Kapitel 1

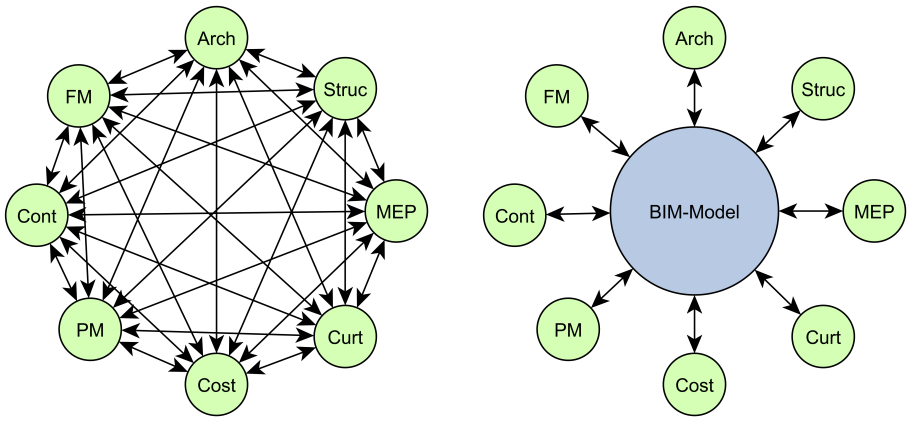
Einführung und Motivation

Nachhaltiges Bauen ist keine Floskel in fachlichen Diskussionen von der Bauindustrie, sondern seit langem in Deutschland durch Regelungen und Kriterien konkretisiert. (BMVBS, 2013) Vernünftige Aussagen und Nachweise sind nicht nur in baulichen Entwurfsphasen erforderlich, sondern auch als Hilfsmittel beim Monitoring, Betriebsmanagement und bei der Entscheidungsfindung für alle Phasen des Lebenszyklus eines Bauwerks wichtig. Moderne Planungsinstrumente wie spezielle Simulationsprogramme werden dazu verwendet, um das energetische, thermische bzw. optische Verhalten eines Gebäudes in zusammenfassenden oder dynamischen Weisen darzustellen.

Der immer stärker werdende Einsatz des Rechners und digitaler Werkzeuge in der Bauindustrie führt zu einer Revolution der Arbeitsweise. Dazu wird ein neuer Ansatz – Building Information Modeling (BIM) – eingeführt, um den Rechnereinsatz in Bauprozesse zu integrieren und gleichzeitig die Beiträge von den Beteiligten aus unterschiedlichen Fachbereichen miteinander zu kombinieren. Die Veränderung der Arbeitsweise wird in Abbildung 1.1 dargestellt. BIM ist ein Prozess von der Planung bis zum Rückbau eines Gebäudes unter Verwendung eines standardisierten, maschinenlesbaren Datenmodells, das alle zuständigen Informationen über den kompletten Lebenszyklus des Gebäudes umfasst. (Eastman *et al.*, 2011) Es gilt als eines der Integrierten Planungsverfahren, die die Zusammenarbeit von allen Projektbeteiligten effizient organisieren und koordinieren können.

Obwohl ein Integriertes Planungsverfahren nur als ein Bewertungskriterium in der Prozessqualität betrachtet wird, das nur mit 10% in den DGNB-Kriterien gewichtet wird (DGNB, 2008), bringt ein BIM-Verfahren zusätzliche Leistungen mit, damit alle Qualitäten in den Nachhaltigkeitbewertungskategorien erhöht werden können:

1. BIM bietet Entwurfshilfen mit Zusatzinformationen für Energie und Umwelt. Durch Dokumentationen in Datenbanken, Analysen und Simulationen können Auswirkungen



(a) Traditionelle Zusammenarbeit **(b)** Zusammenarbeit mit BIM

Abbildung 1.1: BIM im Vergleich mit der traditionellen Zusammenarbeitsweise (nach Nasyrov (2013))

eines Entwurfs auf der ökologischen Seite früh in der Planungsphase erkannt werden, damit eine Optimierung früh durchgeführt werden kann.

2. Auf der ökonomischen Seite können Kosten präziser ermittelt werden. Bei der zentralen Verwendung des Gebäudedatenmodells werden meiste Planungsfehler und darauf zurückführende Mehrkosten vermeidet.

3. Das Gebäudeverhalten und die Qualitäten werden bei den optimierten Planungsprozessen aufgewertet. Eine ganzheitliche Dienstleistung wird dabei auch ermöglicht.

4. Auf der technischen Seite beschleunigt BIM den Einsatz von dem Gebäudeintelligenz, dem computergestützten Gebäudeservice mit Monitoring und dem intelligenten Betriebsmanagementsystem. (Eastman *et al.*, 2011)

Wie Chuck Eastman in seinem Buch schreibt:

> *Perphaps most important is that BIM creates significant opportunity for society at large to achieve more sustainable building construction processes and higher performance facilities with fewer resources and lower risk than can be achieved using traditional practices.* (Eastman *et al.*, 2011)

Er bezeichnet es, wichtigste Eigenschaft von BIM, die Möglichkeit nachhaltigerer Bauprozesse und erhöhter Leistungsfähigkeit mit weniger Ressource und niedrigeren Risiken zu erzielen. Damit kann man davon ausgehen, dass BIM als einer der Wege zum nachhaltigen Bauen zu verstehen ist.

1.1 Motivation

Ein augenfälliger Vorteil, der bei BIM erbracht wird, besteht in der Wechselwirkung zwischen BIM-Modellen und Simulationen. Das BIM-Prinzip erfordert einen nahtlosen automatisierten Datenaustauschprozess zwischen fachlichen Domänen. Bei der traditionellen Datenaustauschmethode wird diese Aufgabe meistens von den beteiligten Fachleuten selbst betätigt. Hierfür werden die für Simulationen benötigten Daten und Informationen von den Architekten, Tragwerks- und TGA-Planern zu den Simulation- und Energieexperten übertragen, und bei diesen per Hand zusammengefasst, interpretiert und in die Simulationsprogramme eingegeben. Dabei stellen die durch die Kommunikationen verursachten Zeit- und Ressourcenaufwand, unerwartete Eingabefehler, sowie interfachliche Missverständnisse die größten Probleme dar.

Schnittstellen sind von vielen Softwareherstellern angeboten, um den Datenaustausch zwischen Modellierung und Simulation zu erleichtern. In machen Simulationsprogrammen kann ein geometrisches Modell im dxf- oder 3ds-Format direkt importiert werden. Andere Informationen wie Materialien und Randbedingungen werden danach hinzugefügt. Bekannt als Speicher der energierelevanten Informationen sind Dateiformate wie gbXML und IDF, die von den meisten Simulationstools wie Ecotect, EnergyPlus und Autodesk Green Building Studio unterstützt werden. Exporter für diese Dateiformate sind in der Modellierungssoftware von größeren Herstellern wie Autodesk und Google angeboten. In Definitionen der zwei Dateiformate werden nicht nur geometrische Daten, sondern auch Daten für Materialien, Umgebungsrandbedingungen, Nutzerverhalten und Betriebsverhalten von Technische Gebäudeausrüstung (TGA) gespeichert. Der Nachteil besteht trotzdem darin, dass diese nicht als allgemeine Datenträger verwendet werden können. Sie enthalten nur eine ausgewählte Teilmenge von Informationen aus dem architektonischen Entwurf und können nicht auf einer BIM-Plattform zurückgeführt werden. Sie fungieren selbst als direkte Schnittstelle zwischen bestimmten Softwareprogrammen und Simulationswerkzeugen, welche die Dateiformate unterstützen. (Nasyrov, 2013) Dabei kann man davon ausgehen, dass wegen der Abhängigkeiten der eingesetzten Software und der gewünschten Interaktionen der Datenaustausch dieser Dateiformate dem vernünftigen BIM-Prinzip widerspricht.

Eine Lösung besteht darin, ein standardisiertes Dateiformat mit umfangreichen Informationen für ein Bauprojekt einzuführen. Hierzu zählt das Industry Foundation Classes (IFC). Ein IFC-Gebäudedatenmodell steht im Mittelpunkt von BIM-Ansätzen und koordiniert die Zusammenarbeit beim Bereitstellen der gleichen und in Echtzeit aktualisierbaren Daten für jede Projektbeteiligten. Die Datenangabe für die Entwickelung der Schnittstelle in dieser Masterarbeit wird im IFC-Dateiformat betrachtet. Der Anlass wird im Kapitel 3 im Detail erläutert.

Auf der anderen Seite der Schnittstelle wird eine Energiesimulationsmethode vorgelegt. Die Modelica-Sprache wird zunehmend in Simulationstätigkeiten eingesetzt. Sie ist eine offen zugängliche und nicht-proprietäre Programmiersprache für Modellierung und Simulationen und ist davon unabhängig, welche Produkte von welchem Softwareherstellern verwendet werden. Darüber hinaus bietet Modelica die Möglichkeiten, den Modellierungs- und Simulationsumfang zu erweitern. Im Vergleich zu den meisten Simulationswerkzeugen im Baubereich, die sich hauptsächlich auf die Gebäude-Seite konzentrieren, ermöglicht die Modelica-Sprache einen Aufbau für komplexe physikalische Systeme, deren Bausteine sich auf mehrere fachliche Domäne beziehen können. Damit kann die Zusammenarbeit von Fachleuten aus der Bau- und Energietechnik verbessert werden. Systemsimulationen (siehe Abbildung 1.2) mit Blöcken wie erneuerbare Energieerzeugung, TGA und Elektro-Mobilität, in denen Gebäude nur eine untergeordnete Rolle spielen, werden stattdessen der Gebäudesimulationen durchgeführt. Das kann den Verbreitungsprozess der Einsätze von neuen Techniken in Bauprojekte nach der Anforderung der Nachhaltigkeit beschleunigen.

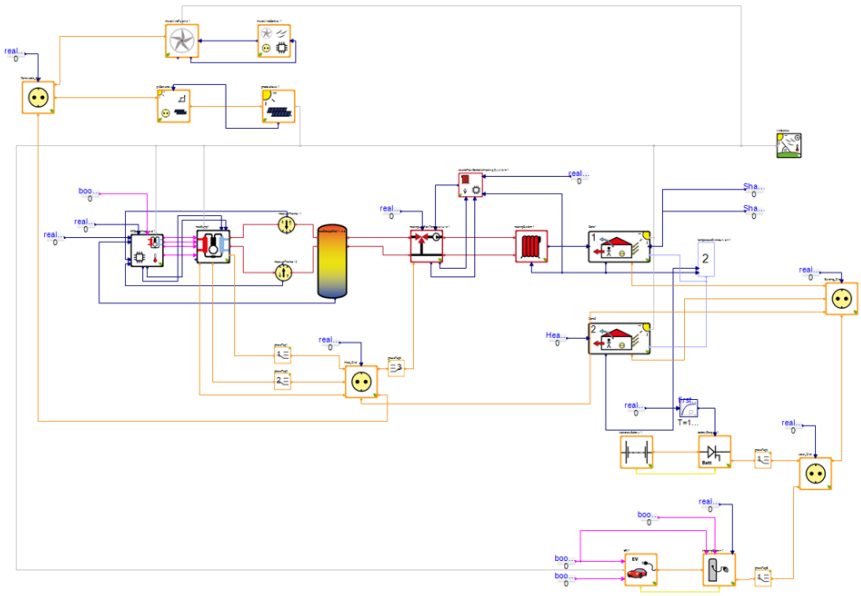


Abbildung 1.2: Aufbau eines Gesamtsystems – *GreenBuilding.Example.ComplexModel* in SimulationX (ITI GmbH, 2013)

1.2 Ziel der Arbeit

Ziel der Masterarbeit ist es, die Möglichkeiten der Einpassung von BIM entsprechenden Gebäudemodellen in Modelica-Simulationsumgebungen zu erforschen. Ausgehend von der Vorarbeit einer selbst entwickelten Modelica-Gebäudebibliothek und der darauf basierenden Analyse der Anforderungen an den Datenaustausch wurde die Methodik und Ansätze zur Implementierung einer Schnittstelle zwischen BIM und Modelica-Simulationen vorbereitet. Ein

prototypisches Werkzeug ist zusammen mit der schriftlichen Ausarbeitung erstellt worden, damit benötigte Daten aus IFC abgerufen werden können und auf Modelica-Codegenerierung bereitstehen. Dieses steht auch für zukünftige Entwicklungen einer für Nutzer anwendbaren Software zur Verfügung.

Für die folgenden Zwecke ist dieses Masterthesis verfasst worden:

1. Der Bedarf von den benötigten Daten für das selbst-entwickelte Modelica-IBPS-Gebäudebibliothek wird hiermit zusammengefasst.
2. Das Thesis fungiert als ein Leitfaden für weitere Implementierungen der Schnittstelle zwischen IFC und Modelica und bietet einen Entwurf der zu implementierenden Komponenten.
3. Eine Bibliothek mit Codestücken wird bereitgestellt, die in der Weiterentwicklung wiederverwendet oder erweitert werden kann.
4. Ein prototypisches Werkzeug wird für weitere Forschungsprojekte angeboten, damit die Datenübertragung von Architekten zu den Modelica-Simulationen vereinfacht werden kann.

1.3 Aufbau der Arbeit

Die Hauptarbeit besteht aus fünf Kapiteln. Nach einer Einführung sollen in Kapitel 2 sollen allgemeine Kenntnisse über Modelica-Simulation und Modelica-Gebäudebibliothek dargestellt werden. Heutige Anwendungen der Modelica in Bauindustrie und im energetischen Bereich werden dabei zunächst erläutert. Anschließend wird vorgestellt, wie die Forscher und Softwarehersteller sich bemühen, in vorhandenen öffentlichen oder kommerziellen Modelica-Simulationsumgebungen Gebäudebibliotheken zu erstellen. Danach wird die selbst entwickelte Gebäudebibliothek, IBPSBuilding-Package, aufgezeigt. Ein Schnittstellenprogramm wird zunächst darauf aufgebaut und kann nach Bedürfnis flexibler eingepasst werden.

Daran anschließend werden in Kapitel 3 Datenanforderungen des Informationsaustausches von IFC-Modellen zur energetischen Simulation in Modelica dargestellt. Neben den erforderlichen Parametern, die direkt in der Modelica-Bibliothek angegeben werden und deren Werte von den IFC-Dateien übertragen werden sollen, wird ein öffentliches Model View Definition (MVD)-Dokument, in dem für architektonischen Entwurf wichtige Energiedaten in IFC-Definitionen aufgezeigt werden, vorgestellt. Zusammenfassend wird gezeigt, welche Vorteil und Schwäche sich zeigen, wenn eine IFC-Datei als Modelleingabe für Energiesimulationen verwendet wird.

Aufbauend auf Kapitel 2 und Kapitel 3 werden in Kapitel 4 Ideen der Implementierung einer Schnittstelle aufgestellt, um eine IFC-Datei in ein Modelica-Modell umzuwandeln. Dabei wird zunächst der prinzipiellen Aufbau der Schnittstelle erläutert. Der zentrale Teil davon wird dann durch Programmierung ermöglicht, damit der Abrufprozess der benötigten Daten von der IFC validiert werden kann. Anschließend wird über unterschiedliche Methoden diskutiert, wie die abgerufenen Daten in Modelica-Code interpretiert werden können. Eine einfache Nutzeroberfläche ist ebenfalls entworfen worden, um einen visuellen Eindruck der Schnittstelle anzubieten.

Im letzten Kapitel wird der Entwicklungsstand der Schnittstelle zusammengefasst. Neben den Abgrenzungen werden hier auch praktische Anwendungs- und Erweiterungsmöglichkeiten für weiteres Forschen und Arbeiten dokumentiert. Erläutert wird am Ende ein Überblick über das Zukunftsbild, in dem die Arbeitsweise des nachhaltigen Bauens und Planens von der verbesserten Interoperabilität zwischen BIM und Modelica-Simulation beeinflusst werden könnte.

Kapitel 2

Aufbau einer Gebäudemodellbibliothek in Modelica

Modelica ist eine offen zugängliche objektorientierte und auf Gleichungen basierende Programmiersprache, die komplexe physikalische Systeme modellieren kann. (Association, 2012). Sie ist im Jahr 1996 bei Modelica Association entwickelt worden. In diesem Arbeit wird ihre aktuelle Version 3.3 verwendet.

Ähnlich wie in anderen objektorientierten Computersprachen werden die Bausteine des Simulationsmodells in Modelica Objekte oder Komponenten genannt. Aus der Beschreibung der Beziehungen zwischen unterschiedlichen Objekten, die aus den Bibliotheken modular ins Modell eingefügt werden und den Zuweisungen ihrer Parameter, wird ein komplexes System aufgebaut. Der Prozess kann mit dem Zusammenbau eines Autos mit Motoren, Getrieben, Fahrgestellen u.a. verglichen werden. Die Datenstruktur eines Objektes (auch Klasse genannt) stellt detaillierte Bestandteilen und Gleichungen innerhalb des Objektes dar. Alle diesen Datenstrukturen werden in Bibliotheken, die in Modelica auch „Packages“ genannt werden, zugeordnet. Die Modelica Association bietet zahlreiche Modelle in verschiedenen fachlichen Domänen an. Diese werden in der vorinstallierten offen zugänglichen Standardbibliothek gespeichert. Mehrere offene Bibliotheken können auf der Webseite http://www.modelica.org herunterladen werden.

Die Modelica-Sprache ist für grafische Umsetzungen in einer Simulationsumgebung vorgesehen. Eine Komponente wird als Symbol dargestellt. Damit kann durch „Drag-and-Drop“ der verschiedenen Symbolen in die Diagrammansicht (siehe Abbildung 1.2) eines Editors ein simulierbares System zusammengebaut werden. Um die Wechselwirkungen zwischen unterschiedlichen Bestandteilen eines Systems zu ermöglichen, wird ein spezifischer Klassen-

typ - „Connector“ eingeführt, damit Anschlussteile wie elektrische Schaltkreise, Eingangs-/Ausgangssignale, oder Angriffsfläche implementiert werden können. Hierdurch können Komponenten miteinander verknüpft und Werte dazwischen übertragen werden.

Zahlreiche Programme für Modelica-Simulationen, entweder kommerziell entwickelte oder offen zugängliche, sind schon in der Praxis eingesetzt (Modelica Association, 2014). Wichtig im deutschen Markt sind Dymola von Dassault Systèmes, SimulationX von der ITI GmbH und OPTIMICA Studio von der Modelon AB. Daneben werden Plattenformen wie JModelica und OpenModelica, die komplett kostenfrei und offen zugänglich sind, auch für Forschungsarbeiten und Zweitentwicklungen häufig verwendet. In dieser Arbeit werden SimulationX und OpenModelica als beispielhafte Werkzeuge ausgewählt.

Modelica wird zunehmend in Industrie eingesetzt. Automobil-Unternehmen wie BMW und Audi verwenden seit lange Modelica, energieeffizienten Autos und verbesserte technische Ausrüstungssysteme zu entwerfen. Laut der Webseite der Modelica Association beträgt die gesamte Kosten für die europäischen Projekte bezüglich Modelica 75 Mio. € zwischen den Jahren 2007-2015. (Modelica Association, 2014) Das stellt einen zukunftsweisenden Weg für Modelica dar.

2.1 Einsatz von Modelica in der Bauindustrie

Es gibt wenig Ansätze, um die Modelica-Sprache in die auf Gebäuden basierenden Simulationen in Bauindustrie einzuführen. Entwicklungen einer Bibliothek, die Gebäude und ihr Verhalten beschreiben kann, existieren in einigen Forschungsprojekten und Geschäftsplänen von Softwareherstellern. Da die Modelica-Sprache spezifisch auf einem hohen Niveau in der Familie der Programmiersprachen steht, kann die Entwicklung einer fachspezifischen Modellbibliothek ohne Probleme von den Fachleuten mit fachlichen Kenntnissen aber nicht von den Informatikern durchgeführt werden. Die Entwicklungen von unterschiedlichen Forschungseinrichtungen orientieren sich an Simulationsergebnissen und damit unterscheiden sich untereinander. Fehlende Standardisierung einer Gebäudebibliothek mit verschiedenen Detaillierungsgraden verhindern einen Datenaustausch und eine vollständige interfachliche Kommunikation. Zwei Beiträge davon werden in diesem Abschnitt vorgestellt:

2.1.1 Modelica Buildings Library

Die offen zugängliche `Buildings`-Package (Modelica Buildings Library) wird vom Lawrence Berkeley National Laboratory (LBNL) seit dem Jahr 2012 entwickelt. Die aktuelle Version 1.5 build 3 wurde im Februar 2014 neu veröffentlicht. Die Bibliothek basiert sich auf dem `Modelica-Fluid`-Package in der Standardbibliothek und ist deswegen kompatibel mit anderen vordefinierten Komponenten. Die Bibliothek wurde in der Dymola-Umgebung validiert.

Um eine robuste Simulationsleistung mit dieser Bibliothek zu erhalten, ist die Installation von Dymola erforderlich. Neben einigen TGA-Systemen und Steuerungsmodulen sind Subpackages für Räume, Bauteile und die physikalischen Wärmetransmissionsprozesse mit enthalten. Der Grundaufbau von einem Raum wird in Abbildung 2.1 dargestellt. (Wetter, 2014)

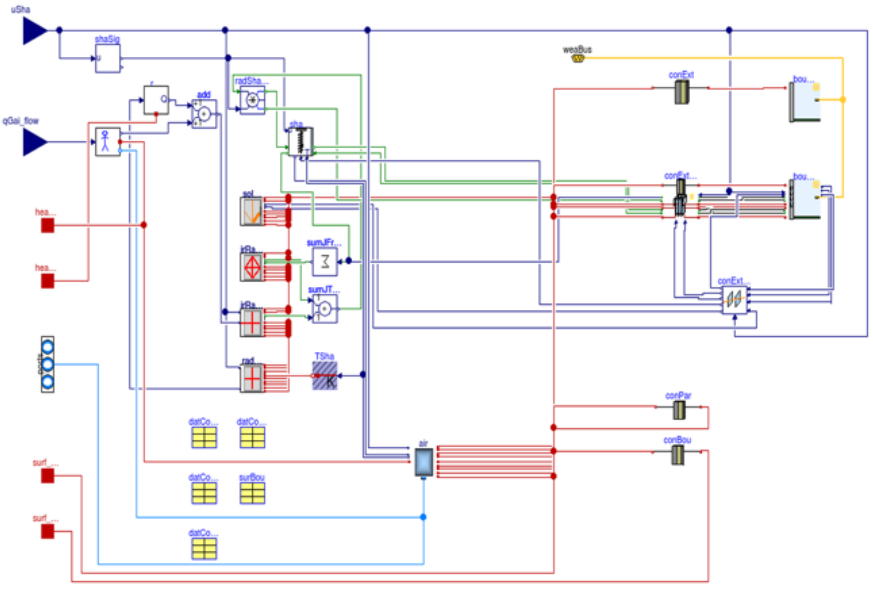


Abbildung 2.1: Aufbau eines Raummodells in LBNL-Modelica Buildings Library (Wetter, 2014)

Das `Buildings`-Package bietet eine gute Möglichkeit, ein Gebäude individuell nach eigenem Bedarf aufzubauen. Ein Nachteil ist die Struktur des Modells und die Eingabemöglichkeiten für seine Parameter, die kompliziert organisiert sind. Die Abhängigkeit von der kommerziellen Software Dymola hindert außerdem Weiterentwicklung und Anwendungen von Dritten.

2.1.2 GreenBuilding-Package in SimulationX

Seit der Version 3.5 von der Software SimulationX ist ein `GreenBuilding`-Package mit eingeschlossen. Umfangreiche neue Techniken für TGA und erneuerbare Energieerzeugung wurde darin in Modulen beschrieben. Für den Aufbau eines Gebäudes werden Modelle im Subpackage `Building` verwendet. Im Vergleich zu der „Modelica Buildings Library“ vom LBNL besitzen die Definitionen von Räumen (od. Zonen) keinen systematischen Aufbau aus detaillierten Submodellen, sondern werden als ein unteilbares komplexes Modell programmiert. Die physikalischen Prozesse werden als „Blackbox“ in einem Raum definiert und können schwierig validiert werden. Die Eingabe des Aufbaus eines Raums ist nur beschränkt durch eine Eingabemaske (siehe Abbildung 2.2) ermöglicht. Darin ist die Anzahl der Raumgrenzen auf 9 begrenzt und eine komplizierte Raumform kann nicht interpretiert werden. Außerdem sind die Rechenverfahren der physikalischen Prozesse vereinfacht. Für Wärmetransmission durch Bauteile sind nur U-Werte aber keine Eigenschaften von einzelnen Materialschichten möglich.

Das widerspricht dem Grundprinzip von einer dynamischen Energiesimulation, das im Buch von Clarke (2001) erwähnt wird.

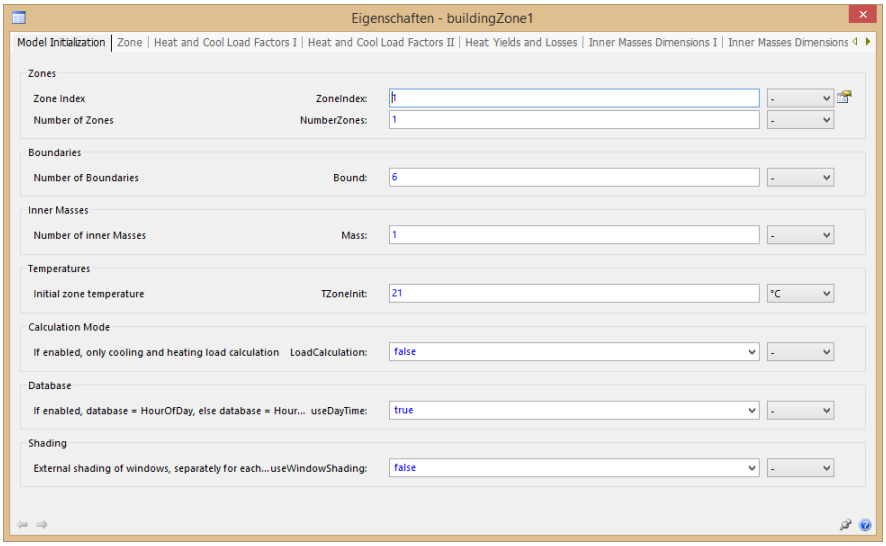


Abbildung 2.2: Die Eingabemaske für einer Zone aus dem SimulationX-*GreenBuilding*-Package

2.2 IBPSBuilding-Package

Im Rahmen einer Lehrveranstaltung an der TU München „Implementing a Building Performance Simulation (IBPS)“ wurde eine eigenen Modelica-Gebäudebibliothek - „IBPSBuilding-Package“ konzipiert und verwendet. In dieser Lehrveranstaltung wird den Studierenden beigebracht, wie man mit Hilfe von Rechnern die Wärmetransmissionsprozesse für Gebäude systematisch aufbauen kann, um thermisches Gebäudeverhalten zu simulieren. Die erste Version dieser Bibliothek wurde im Jahr 2012 nach dem Konzept von PD Dr.-Ing. Christoph van Treeck in den Vorlesungen (van Treeck, 2011) in OpenModelica-Editor umgesetzt. Der Aufbau basiert auf der Methodik nach Clarke (2001)). Um eine bessere grafische Darstellung und dadurch eine Überschaubarkeit für die Vorstellung im Seminar und Weiterentwicklungen zu erreichen, wurde die OpenModelica-Bibliothek in SimulationX interpretiert. Der Hauptunterschied zwischen den beiden Umgebungen für High-Level-Entwickler[1] besteht in der Übersetzungsmethoden der „annotation“-Sätze zu Grafiken. Vernachlässigt man diese Zusatzinformationen und unterschiedliche Kompilierungs-/Simulationsleistungen sind die Strukturen der Modelle innerhalb der beiden Bibliotheken plattformunabhängig. Hierbei werden in den folgenden Abschnitten Codes und Bilder in den beiden Umgebungen gemischt genutzt, damit die Arbeit möglichst verständlich vorgestellt werden kann.

[1] Fachleute, die Modelica-Sätze schreiben oder Modelle durch „Drag-and-Drop“ zusammenstellen, um Systemsimulationen direkt laufen zu lassen

Wie andere oben genannte Modelica-Implementierungen für Gebäuden wurde das Konzept vom IBPSBuilding-Package auch ausgehend von der Besonderheit des Modelica-Arbeitsprozesses entwickelt. Durch das Zusammenschließen der Connectors von verschiedenen Modulen können alle Randbedingungen, Bauelemente und darauf bezogene physikalische Energieübertragungsprozesse modellieren werden. Dabei kann jede detaillierte Angabe modifizieren werden. Das Package untergliedert sich in sieben erarbeitete Subpackages: Interfaces, HeatTransferUnits, AmbientConditions, Constructions, Zones, MaterialLibrary und Functions. In den letzten beiden ist kein instanziierbares Modell, sondern nur die für die Simulation benötigten Daten und Funktionen vorhanden.

Subpackage „Interfaces“

Im Subpackage `Interfaces` werden alle grundlegenden Funktionseinheiten gespeichert. Unter dem Begriff Funktionseinheiten versteht man drei Arten von Modellen: Connectors; Partialmodelle, die nur ein Rahmenwerk mit Connectors und Zusatzinformationen wie grafische Darstellungen enthält; und reine logische Modelle wie Connector-Duplikatoren. Diese sind keine Abbildungen von realen Objekten in der physikalischer Welt, sondern dienen als Hilfsteile, die in anderen komplexen Modellen genutzt werden.

Zwei beispielhafte wichtige Connectors sind die beiden `HeatPorts`. (Einer dient als Eingangspunkt des Wärmestroms, und der andere dient als Ausgangspunkt.) Jeder `HeatPort` hat zwei Variablen: eine Temperatur und einen Wärmestrom. Die zusammenschließenden Connectors haben gleiche Temperaturen und einen Ausgleich von Wärmen, d.h. die Summe ihrer Wärmeströme ist gleich null. Mit dieser Methode kann die Energie zwischen Modulen durchfließen. Die Connectors sind vordefiniert und aus der Standardbibliothek abgeleitet, damit die IBPS-Modelle auch mit den standardisierten oder von Dritten entwickelten Packages gekoppelt werden können.

Subpackage „HeatTransferUnits“

In diesem Subpackage werden physikalische Wärmeübertragungsprozesse definiert. Hierzu sind Komponenten wie Wärmeleitung, Konvektion, kurz- und langwellige Strahlung, Wärmespeicherung sowie Infiltration mit eingeschlossen. Beziehungen zwischen Parametern und Variablen wie Wärmeströme, Temperaturen, Zeit und ihre Ableitungen werden durch Gleichungen festgelegt. In Energiesimulation bestehen die baulichen Komponenten ausschließlich aus diesen Bausteine.

Subpackage „AmbientConditions“

Im Kern des Subpackages steht das Modell `Ambience`, das Wetterdaten von externen Tabellen einlesen, die Daten verarbeiten und die durch angepasste Connectors anderen Modellen weitergeben kann. Die Tabellen mit Wetterdaten für verschiedene Standorte können von den meisten Klimawerkzeugen wie Meteonorm und WeatherTools Ecotect exportiert werden. Klimadatenbanken und Wetterstationen bieten auch Dateien mit CSV-und TMY-Formaten zum Download an. Auf der beigefügten CD befindet sich ein einfaches Excel-VBA-Macro, das diese Tabellen in ein Modelica-einlesbares Dateiformat umwandelt. In der SimulationX-Version steht die Möglichkeit, als Datenquelle das `Ambience`-Modell im vorinstallierten „GreenBuilding“-Package zusammen mit SimulationX vom ITI-Unternehmen zu benutzen.

Subpackage „Constructions“

Das Subpackage `Constructions` enthält die wichtigsten Bestandteile mit baulichen Bedeutungen. Diese Modelle sind als Blackboxen und als Grundsteine für allgemeine Endnutzer vorgesehen, indem physikalische Prozesse darin verkapselt sind. Zwei Beispiele werden hierbei aufgezeigt: Abbildung 2.3 stellt dar, wie eine einfache opake Materialschicht aufgebaut wird. Es gibt drei äußere Anschlüssen, jeweils einen für die Innen- und Außenoberflächen, sowie einen zusätzlichen Wärmezufuhr/-abfuhr, der direkt zum Schichtenkern steht. Daneben gibt es in einer `ConstructionLayer` noch zwei Wärmeleitungsmodelle und einen Wärmespeicher, dessen Eigenschaften durch eingegebene Parameter ausgerechnet werden können. Die Komponenten enthalten dafür eine Parameterliste vom Typ `ParameterSheet.Layer`. Um Materialeigenschaften einzugeben, wird der Modelltyp für das Objekt `mat` umgetauscht. Als Alternative sind die Listen im Subpackage `MaterialLibrary` verfügbar. Es ist auch möglich, die Werte der thermischen Eigenschaften von den im Package vorhandenen Materialien zu modifizieren, oder ein komplett neuer Materialtyp zu erstellen.

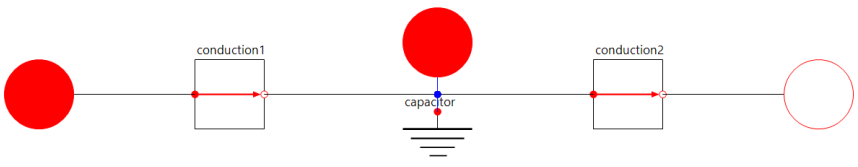


Abbildung 2.3: Aufbau des Modells „ConstructionLayer“

Viel komplizierter wird das Modell für Bauelemente wie in Abbildung 2.4 gezeigt gestaltet. Die interne Struktur wird in diesem Text nicht im Detail erläutert.

Durch Eingaben der Parameterwerte können die Modelle in Einzelfall in spezifische Bauteile instanziiert werden. Als ein Beispiel wird ein Instanziierungsprozess des `ConstructionComponent` folgend dargestellt. In Tabelle 2.1 aufgelistete Parameter und der darin definierte Aufbau

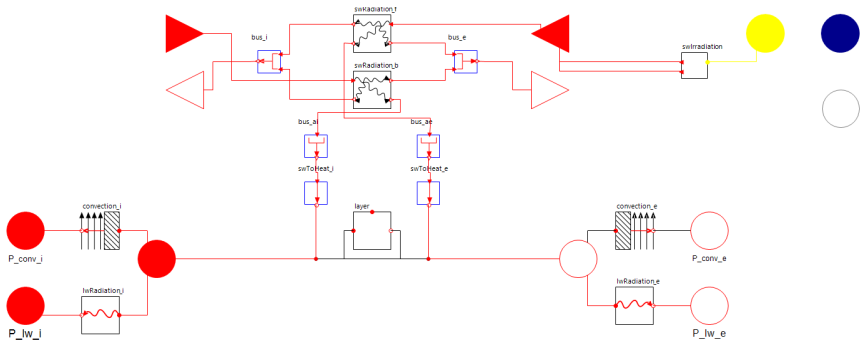


Abbildung 2.4: Aufbau des Modells „ConstructionComponent“

werden in Modelica-Code wie Listing 2.1 interpretiert. Dabei entsteht eine allgemeine Parameterliste mit dem Namen „`parc`“, in der alle auf dem Bauteil selbst bezogenen Parameter wie Geometrie und Orientierung deklariert werden und ein paar Parameterlisten vom Typ `ParameterSheet.Layer`, die aus Materialien, Schichtdicke und dem Anteil der Unterkonstruktionen bestehen. Die Parameterlisten nehmen Eingabedaten auf und geben die den Submodellen(Bauschichten, Konvektionsmodelle, Solarstrahlungsaufteiler, etc.) weiter.

Tabelle 2.1: Parameter der Beispielswand „extWallNorth“

Orientierung	Nord (Azimuthwinkel = 0)
Neigungswinkel	90° (vorgegeben)
Wandtyp	Außenwand (vorgegeben)
Höhe	3 m
Länge	5 m
eingebettete Fenster	2 St.
Aufbau von Innen nach Außen	2 cm Innenputz aus Gips 20 cm Stahlbeton 10 cm Wärmedämmverbundsystem (WDV)

Listing 2.1: Instanziieren der „ConstructionComponent“ in die Wand „extWallNorth“

```
1  Constructions.ConstructionComponent extWallNorth(
2      parc(
3          azimuth=0,
4          height=3,
5          length=5,
6          nLayers=3,
7          nWindows=2),
8      parLayer1(
9          redeclare replaceable
               IBPSBuilding.MaterialLibrary.BuildingMaterials.Plaster_gips mat,
               thickness=0.02),
10     parLayer2(
```

```
11          redeclare replaceable
                IBPSBuilding.MaterialLibrary.BuildingMaterials.Concrete mat,
                thickness=0.2),
12      parLayer3(
13          redeclare replaceable
                IBPSBuilding.MaterialLibrary.BuildingMaterials.InsulationSystem
                mat, thickness=0.1));
```

Subpackage „Zones“

Ein Raum dient als eine Grundeinheit in Energiesimulationen. In Gebäudesimulationen werden die Räumen innerhalb des Gebäudes separat betrachtet, damit die Ergebnisse jeweils ausgegeben werden können. Die wichtigste Aufgabe für Modellierung eines simulierbaren Gebäudesystems besteht somit darin, Räume aufzubauen, ihre Verbindungen zu definieren, und Informationen für TGA und Nutzerverhalten in jedem Raum einzugeben. Hierbei steht in diesem Subpackage eine Vorlage – `ZoneEmpty` für Ableitungen zur Verfügung. Benötigte Anschlüsse für Wetterdaten und Wärmezufuhr/-abfuhr, sowie von Bauteilen unabhängige Komponenten wie Infiltration und Raumluft sind darin vorgegeben. Beim Ableiten dieses Modells bekommt man neue modifizierbare Zonenmodelle, in die die raumabschließenden Bauteilen, TGA und andere Module nach Bedarf durch „Drag-and-Drop“ oder durch Kodierung eingefügt werden können. Ein beispielhaftes Raummodell wird unter Abbildung 2.5 dargestellt. Es gibt in diesem Raum 6 Bauteilen(4 Wände, 1 Boden und 1 Decke) und ein Fenster. Die eingehenden Strahlungen werden nach Flächen und Absorptionskoeffizienten der Innenoberflächen auf alle Bauteile anteilig verteilt.

2.3 Zusammenfassung

Ein simulierbares Modell mit dem IBPSBuilding-Package wird nach dem Prinzip – Projekt → Zonen → Bauteile → Materialschichten → Materialien aufgebaut. Die Aufgabe für die Nutzer auf einem hohen Niveau (d. h. die für Simulationen verantwortlichen Fachleute im Baubereich) besteht darin, die benötigten Zonenmodelle nach der Vorlage `ZoneEmpty` zusammenzubauen und sie ins Projekt im Editor von einer Modelica-Umgebung einzufügen. Mehrere Zonen werden durch Anschlüsse miteinander und auch mit dem Modell `Ambience` mit Lageinformationen und Wetterdaten verknüpft. Zusätzliche Anschlüsse, die von den `Connectors` aus der Standardbibliothek abgeleitet werden, werden auch in Zonenmodellen angeboten, damit andere Techniksysteme daran angeschlossen werden können.

Testen und Fehlersuche für das IBPSBuilding-Package wird in der Umgebung von OpenModelica und SimulationX durchgeführt, damit die Kompatibilität mit unterschiedlichen Plattfor-

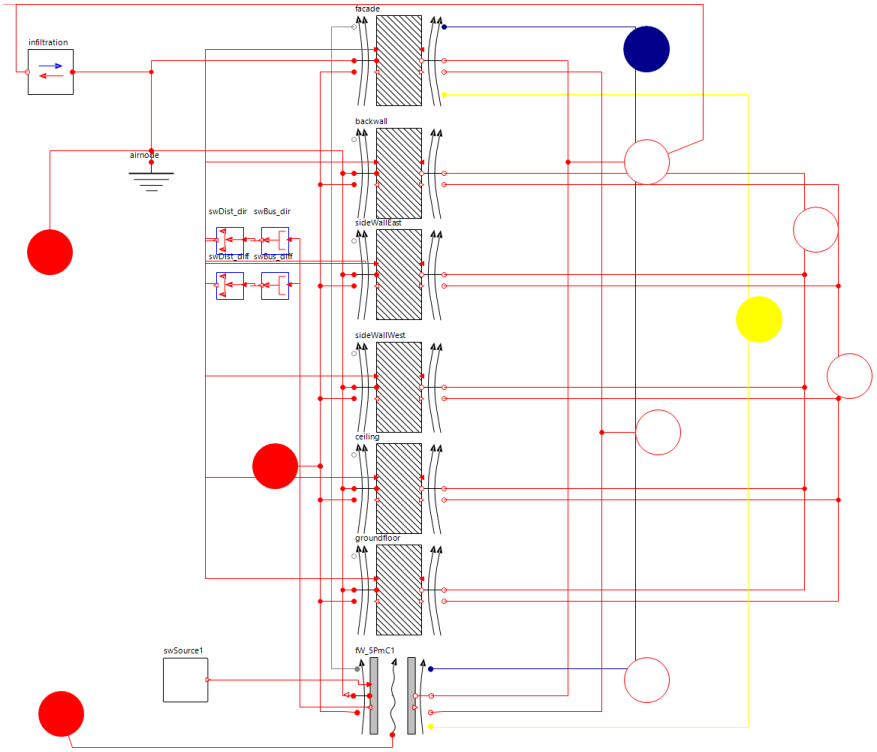


Abbildung 2.5: Aufbau eines Beispielraums

men überprüft werden kann. Durch eine Validierung sollen die Ergebnisse aus diesem Package mit denen aus „Modelica Buildings Library“ von LBNL verglichen werden.

Bis jetzt werden die Anschlüsse zwischen Nebenräumen noch nicht implementiert. Im zukünftigen Erweiterungsplan stehen auch Vorbauten und Nutzerverhalten. Mehr Module und Eigenschaften für Ausführung einer Licht- oder Komfortsimulation können hinzugefügt werden. Um verschiedenen Detaillierungsgraden (engl. Level of Detail (LOD)) von Bauprojekten gerecht zu werden, soll die Bibliothek auf mehreren Ebenen erweitert und modifiziert werden, damit passende Leistungen und Simulationsgeschwindigkeiten für Modelle nicht nur in einem großen Maßstab (z. B. Energiesimulation eines Quartiers) sondern auch in einem kleinen Maßstab (z. B. Simulationstests für neue Materialien oder Konstruktionen) ausgewählt werden können.

In diesem Kapitel wurde die Form von Zieldaten dargestellt. Im nächsten Kapitel soll nun eine vorhandene Datenangabe, die von Architekten aus einer BIM-Plattform für Simulationen bereitgestellt werden, ausführlich erklärt werden.

Kapitel 3

Benötigte Daten aus IFC-Modellen für Energiesimulationen

Wie im Kapitel 1.1 erläutert, spielt das IFC im BIM-Ansatz eine entscheidende Rolle. Dieses öffentlich zugängliche Dateiformat wird seit dem Jahr 1995 von buildingSMART International (bSI) (früher als IAI bekannt) entwickelt und unterstützt. IFC wird heutzutage als das einzige nicht-proprietäre und gut entwickelte Produktdatenmodell für Gebäuden anerkannt. (Eastman *et al.*, 2011) Es wird bisher nicht nur als ein Standard des Datenaustausches von der Bauindustrie in verschiedenen Staaten (z. B. in den USA und Singapur) verwendet, sondern auch mit ISO als ISO 16739 zertifiziert. (buildingSMART International, 2014)

IFC wird in der EXPRESS-Sprache definiert, damit das den „ISO-Standard for the Exchange of Product model data (STEP)“ entspricht. Betrachtet man die bestehende Standardisierung von der Datenumfassung der Industrieprodukte durch STEP, v. a. den in Deutschland genormten Datenaustausch im Bereich der TGA, kann IFC damit sehr kompatibel sein und dadurch die Interoperabilität zwischen mehreren Fachbereichen ermöglichen.

IFC ist als ein allgemeiner Datencontainer für ein Bauprojekt vorgesehen. Das Ziel des IFCs ist, ein erweiterbares Rahmenwerk anzubieten, um möglichst vollständige Informationen für ein Bauprojekt im ganzen Lebenszyklus zu umfassen. (Eastman *et al.*, 2011) Das heißt, dass nicht alle in IFC gespeicherten Informationen sind für eine bestimmte fachliche Domäne oder für eine konkrete Aufgabe benötigt werden. In diesem Kapitel wird darüber diskutiert, welche Daten für eine Energiesimulation eines Gebäudes notwendig sind, und wie die in einer IFC-Datei abgespeichert werden.

3.1 Grundlagen und Terminologie

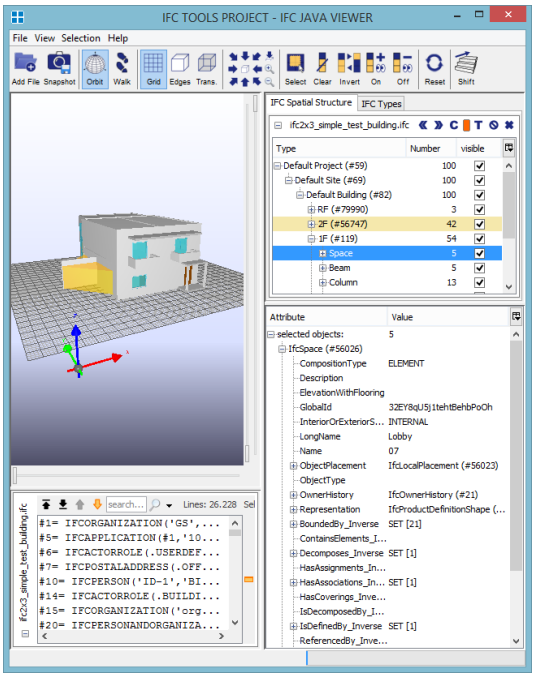
Um den Abrufprozess von benötigten Daten aus einem IFC-Modell zu erklären, wird die Datenstruktur von IFC in diesem Kapitel analysiert.

Eine IFC-Datei ist ein objektorientiertes Datenmodell, in dessen Hauptteil Reihen von Objekten, die ein Bauprojekt darstellen, aufgelistet werden. Der Inhalt von einem beispielhaften IFC-Modell wird in Abbildung 3.1 gezeigt. Abbildung 3.1a stellt die ersten Zeilen eines Quellcodes dar. Jede der Zeilen steht für ein Element, das zum Bauprojekt gehört. Ein Element ist nicht nur ein materielles Bauteil wie eine Wand oder ein Fenster, sondern auch alle anderen relevanten Informationen werden hier repräsentiert. Hierzu zählen unter anderem: Metadaten wie Versionen, Erstellungszeit, Kontakt der Eigentümer, usw.; Geometrie in verschiedenen Formen von Repräsentationen; Eigenschaften, die in *Property-sets (P-sets)* geschrieben werden; Beziehungen zwischen mehreren Objekten; grundlegende Aufzählungen und Größen, die bei anderen Elementen abgefragt werden können; etc.

Viele Werkzeuge können ein IFC-Modell hierarchisch strukturiert bzw. anschaulich in Dreidimensional (3D) darstellen. In dieser Masterarbeit wird ein Toolkit von der „IFC TOOLS Project“ verwendet. In diesem Project wird ein Rahmenwerk für das Zugreifen und Visualisieren IFCs sowie Möglichkeiten zur Weiterentwicklung IFC-unterstützender Applikationen angeboten (Tauscher, 2013). Dazu gehört ein IFC-WebGL-Viewer und ein IFC-Java-Viewer, die eine Strukturansicht wie Abbildung 3.1b bieten können.

```
DATA;
#1= IFCORGANIZATION($,'Autodesk Revit 2014 (DEU)',$,$,$);
#5= IFCAPPLICATION(#1,'2014','Autodesk Revit 2014 (DEU)','Revit');
#6= IFCCARTESIANPOINT((0.,0.,0.));
#9= IFCCARTESIANPOINT((0.,0.));
#11= IFCDIRECTION((1.,0.,0.));
#13= IFCDIRECTION((-1.,0.,0.));
#15= IFCDIRECTION((0.,1.,0.));
#17= IFCDIRECTION((0.,-1.,0.));
#19= IFCDIRECTION((0.,0.,1.));
#21= IFCDIRECTION((0.,0.,-1.));
#23= IFCDIRECTION((1.,0.));
#25= IFCDIRECTION((-1.,0.));
#27= IFCDIRECTION((0.,1.));
#29= IFCDIRECTION((0.,-1.));
#31= IFCAXIS2PLACEMENT3D(#6,$,$);
#32= IFCLOCALPLACEMENT(#5828,#31);
#35= IFCPERSON($,'Hua','Shan',$,$,$,$,$);
#37= IFCORGANIZATION($,'','',$,$);
#38= IFCPERSONANDORGANIZATION(#35,#37,$);
#41= IFCOWNERHISTORY(#38,#5,$,.NOCHANGE.,$,$,$,1370296635);
#42= IFCSIUNIT(*,.LENGTHUNIT.,$,.METRE.);
#43= IFCSIUNIT(*,.AREAUNIT.,$,.SQUARE_METRE.);
#44= IFCSIUNIT(*,.VOLUMEUNIT.,$,.CUBIC_METRE.);
#45= IFCSIUNIT(*,.PLANEANGLEUNIT.,$,.RADIAN.);
#46= IFCDIMENSIONALEXPONENTS(0,0,0,0,0,0,0);
#47= IFCMEASUREWITHUNIT(IFCRATIOMEASURE(0.0174532925199433),#45);
#48= IFCCONVERSIONBASEDUNIT(#46,.PLANEANGLEUNIT.,'DEGREE',#47);
#49= IFCSIUNIT(*,.MASSUNIT.,.KILO.,.GRAM.);
#50= IFCSIUNIT(*,.TIMEUNIT.,$,.SECOND.);
#51= IFCSIUNIT(*,.THERMODYNAMICTEMPERATUREUNIT.,$,.KELVIN.);
#52= IFCDERIVEDUNITELEMENT(#49,1);
#53= IFCDERIVEDUNITELEMENT(#51,-1);
#54= IFCDERIVEDUNITELEMENT(#50,-3);
#55= IFCDERIVEDUNIT((#52,#53,#54),.THERMALTRANSMITTANCEUNIT.,$);
#57= IFCDERIVEDUNITELEMENT(#42,3);
#58= IFCDERIVEDUNITELEMENT(#50,-1);
#59= IFCDERIVEDUNIT((#57,#58),.VOLUMETRICFLOWRATEUNIT.,$);
```

(a) Codezeilen



(b) Strukturansicht in Java-Viewer

Abbildung 3.1: Ein Beispiel von einem IFC-Modell

Für die weitere Erläuterung in diesem und den nächsten Kapiteln ist es nötig, folgende Begriffe zu erklären (nachfolgend nach buildingSMART International, 2013):

Eine **Entität** ist, wie eine Klasse in der objektorientierten Programmierung, ein abstraktes Modell, das als Bauplan für die Abbildung von realen Objekten dient und ihre Eigenschaften beschreibt. Im Vergleich zu einer Klasse wird das Datenverhalten(Methoden) in einer Entität nicht erwähnt. Je nach Kontext wird ein Entität **Objekttyp** bezeichnet.

Ein **Attribut** ist ein Absatz, der innerhalb einer Entität definiert wird. Die Attribute können beschreibend oder ihre Entität identifizierend sein. Ein **Objekt** wird nach der Struktur von einer Entität durch Zuweisungen der Werte zu allen ihren Attributen ausgeprägt. Ein solches Objekt wird somit auch **Instanz** oder **Exemplar** dieser Entität genannt. Identifizierende Attribute heißen auch **Identifikatoren**, bei denen ein Objekt eindeutig von anderen unterschieden werden kann. Die Identifikation eines Objektes von einem IFC-Modell wird durch „**Globally Unique ID (GUID)**“ ermöglicht. In der Implementierung ist es auch praktisch, die Zeilennummer abzurufen, damit ein bestimmtes Objekt ausgelesen werden kann.

„**Property**“ und „**Property-set (P-set)**“ machen das Rahmenwerk von IFC flexibler. Mit später Bindung wird ein „Property“ als ein Instanz dynamisch definiert. Eine „Property“ ist eine Dateneinheit, die eine wichtige Rolle in der Implementierung spielt.

Jedes Objekt, das eine Zeile vom IFC-Code darstellt, wird von einem bestimmten Objekttyp durch Zuweisung der Werte oder Referenzen von anderen Objekten zu seinen Attributen erzeugt. Die Entitäten/Objekttypen sind konzeptionell in der IFC-Dokumentation hierarchisch in den Datenschemen auf vier Ebenen organisiert. (buildingSMART International, 2013) Infolgedessen gliedern sich die Informationen eines Objektes normalerweise in eine komplexe tiefe Baumstruktur auf, die zu Schwierigkeiten bei Datenanalyse- und Datenabrufprozessen führen kann.

Die unterste Hierarchiestufe ist der **Ressource Layer**, in dem alle Entitäten für grundlegende Daten vorhanden sind. Beispielsweise werden P-sets, einfache Menge und Koordinaten von Punkten und Linien definiert. Diese haben keine eigene Identität und unterscheiden sich nur nach Werten von anderen. Im Datenabrufprozess dienen diese Entitäten auch als Endknoten von Durchsuchungen.

Die Entitäten in der **Core Layer** sind als allgemeine abstrakte Modelle abgebildet, damit mehrere Begriffe auf den höheren Ebenen davon abgeleitet werden können. Hierbei werden die häufig verwendeten Attribute wie GUIDs und Metadaten beschrieben.

Interoperability Layer wird als die wichtigste Ebene für die weitere Implementierung betrachtet, denn er umfasst alle Bauelemente, die von allen Fachbereichen gemeinsam genutzt werden. Im Vergleich dazu steht der **Domain Layer**, in dem die fachspezifischen Elemente aufgelistet werden. Detaillierte dynamische Energiesimulationen erfordert umfangreiche In-

formationen aus mehr als einer Domäne, die spezialisiert sind und nicht zu einem gemeinsam genutzten Schema gehören. Deshalb ist es auch nötig, die Entitäten auf der obersten Ebene zu berücksichtigen.

In den folgenden Abschnitten wird analysiert, welche konkrete Daten, die von Energiesimulationen benötigt werden, aus IFC-Modellen abgerufen werden können. IFC wird auch für die Speicherung der Wetterdaten, des Nutzerverhaltens, der Steuerungsstrategien und anderen Betriebsdaten vorgesehen. Entitäten, Aufzählungen von Variablen und P-sets auf der obersten Ebenen sind dafür verfügbar. Diese sind allerdings von den häufig in der Praxis eingesetzten BIM-Plattformen ungenügend unterstützt. Beim Aufbau eines flexiblen Energiemanagementsystems in einer Modelica-Umgebung werden solche Informationen häufiger direkt eingegeben oder von anderen Funktionsmodulen ausgefüllt. Betrachtet man die fünf von Simulationen erforderten Datenbereiche (Nasyrov, 2013) beschränkt sich die Analyse und Implementierung in dieser Masterarbeit nur auf Gebäudedaten. Dabei stehen drei Aspekten: Geometrie und Orientierung, Materialien und physikalische Eigenschaften, sowie Verbindungen zwischen Räumen und Bauteilen im Mittelpunkt. Daten für Standortbedingungen, Gebäudebetrieb und TGA sind hier vom direkten Datenaustausch zwischen IFC und Modelica ausgeschlossen.

3.2 IBPSBuilding-Parameter

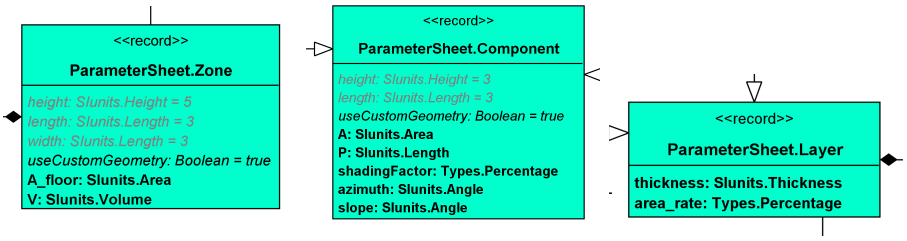
Für die Durchführung einer Modelica-Simulation wird es vorausgesetzt, dass alle Parameter der Elemente im Simulationsmodell gesetzt werden. Durch eine entsprechende Schnittstelle wird dieser Prozess automatisch ausgeführt. Im Idealfall sollten alle Parameterwerte auf eine IFC-Datei, die aus einer BIM-Plattform exportiert wird, zurückgegriffen werden. Im Anhang A zeigt Abbildung A.1 nicht nur die Topologie von benötigten Modellklassen aus IBPSBuilding-Package(Subpackage `Interfaces`, `HeatTransferUnits` und `AmbientConditions` werden vernachlässigt), sondern auch dafür erforderliche Parameter. In den folgenden Abschnitten werden sie in den oben genannten drei Aspekten zusammengefasst:

3.2.1 Geometrie und Orientierung

Für Zonen und Bauteilen steht die Möglichkeit im IBPSBuilding-Package, die Bemessungen der Räumen und Bauteilen wie Höhe, Länge und Breite manuell einzugeben, damit die Flächen, Volumen und Umfänge innerhalb des Simulationsprozesses automatisch berechnet werden können. Das wird beim Import der Parameter von einer IFC-Datei vernachlässigt. Dazu soll der Boolean-Wert `useCustomGeometry` immer wahr fixiert werden. Die grau dargestellten Parameter in Abbildung 3.2 haben keinen mehr Einfluss auf die Modellierung und Simulation. Im Gegensatz dazu sind die fett geschriebenen zwingend von der IFC-Datei erforderlich. Eine Materialschicht verlangt eine Schichtdicke und optional einen anderen Parameter

mit dem Namen `area_rate`, der nur einen Anteil einer Schicht mit einem speziellen Material ausgefüllt, z. B. die Wärmedämmung zwischen Holzunterkonstruktionen, beschreibt.

Die Orientierung von einem Bauteil wird nach dem Azimuthwinkel(`azimuth`) und dem Neigungswinkel(`slope`) ausgerechnet. Ob es eine Wand oder eine Decke ist, definiert der Neigungswinkel. Diese Unterscheidung beeinflusst die dynamische Rechenverfahren von Konvektionskoeffizienten.



(a) Raum **(b)** Bauteil **(c)** Materialschicht

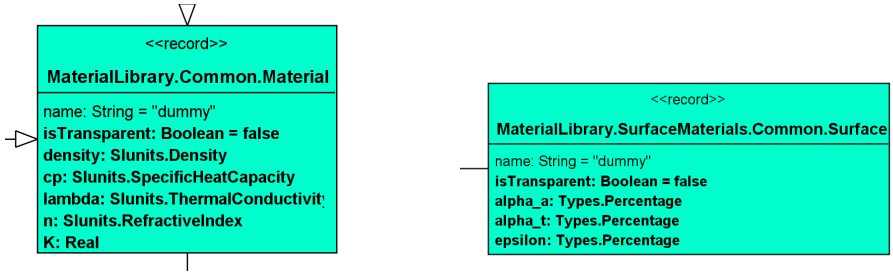
Abbildung 3.2: 3 Beipiele von ParameterSheets, die geometrische Daten beinhalten

3.2.2 Materialeigenschaften

Jede Materialschicht besitzt ein `mat`, dessen Typ von `MaterialLibrary.Common.Material` abgeleitet wird. In den Modelldefinitionen wird das Submodell `mat` erst im allgemeinen Typ deklariert und bei der Instanziierung durch Redeklaration spezialisiert. Beim Import einer IFC-Datei können die Parameter innerhalb `mat` direkt bei der Schnittstelle gesetzt werden. Hiermit stellt Abbildung 3.3a die Parameter für Materialeigenschaften dar. Neben den normalen wie Massendichte(`density`), spezifische Wärmespeicherfähigkeit(`cp`) und Wärmeleitfähigkeit(`lambda`) sind zwei abhängige Parameter – Brechungsindex(`n`) und Extinktionskoeffizient(`K`) erforderlich, wenn der Boolean-Wert `isTransparent` als wahr gesetzt wird. Ebenso stehen zwei Datensätze vom Typ in `MaterialLibrary.SurfaceMaterials` (Abbildung 3.3b) in jedem Bauelement jeweils für die beiden Oberflächen. Damit kann der Absorptionsgrad `alpha_a` und der Transmissionsgrad `alpha_t`, mit denen der Reflexionsgrad `alpha_r` auch ausgerechnet werden können, sowie die Emissivität `epsilon` für langwellige Strahlungen eingegeben werden.

3.2.3 Topologie

Die Topologie von einem simulierbaren Modellen stellt die Assoziationen zwischen beteiligten Komponenten dar. Betrachtet man den Datenaustausch für ein Gebäude werden zwei Arten von Beziehungen berücksichtigt: Komposition und Anschlussverbindung. Bei einer Komposition dient ein Objekt von einem Typ als ein Bestandteil von einem anderen Typ. Diese Zugehörigkeit soll von einer IFC-Datei in Modelica wie Abbildung 3.4 gezeigt interpretiert



(a) Materialeigenschaften **(b)** Oberflächeeigenschaften

Abbildung 3.3: ParameterSheets für Materialeigenschaften

werden. Auf den Ebenen von Bauteilen und Materialschichten sind die Aufbauten eines Modells in der Modelica-Bibliothek vordefiniert und voll parametrisiert. Das heißt, dass keine zusätzliche Informationen darüber, wie z. B. die Verbindungen zwischen verschiedenen Materialschichten und die Beziehungen zwischen Schichten und Wandoberflächen, von IFC-Dateien importiert werden müssen. Zusätzliche Parameter wie `nLayers` und `nWindows` werden so gesetzt, dass die Anzahl von Materialschichten oder die von den Anschlüssen zu den Fenstern (sog. Größe der Arrays) vor der Kompilierung angegeben werden soll.

Die Verbindungen durch Modelica-"`connect()`"-Gleichungen innerhalb eines Bauelements sind vorgegeben. Bei der Instanziierung eines Bauelements werden zusätzliche Gleichungen in der Code-Generierungsphase eines Zonenmodells benötigt, um die Verbindungen von Bauelementen zu dem Raumluftknoten, den Strahlungsverteilern und den Außenanschlüssen zu erstellen. Die Parameter `isExternal` und `connectToAdiabat` steuern die Generierung von diesen Gleichungen und beschreiben die Randbedingungen eines Bauelements, d. h., ob es an die Außenumgebung, an eine fixe Temperatur, oder an den Knoten von einem Nebenraum angeschlossen werden soll.

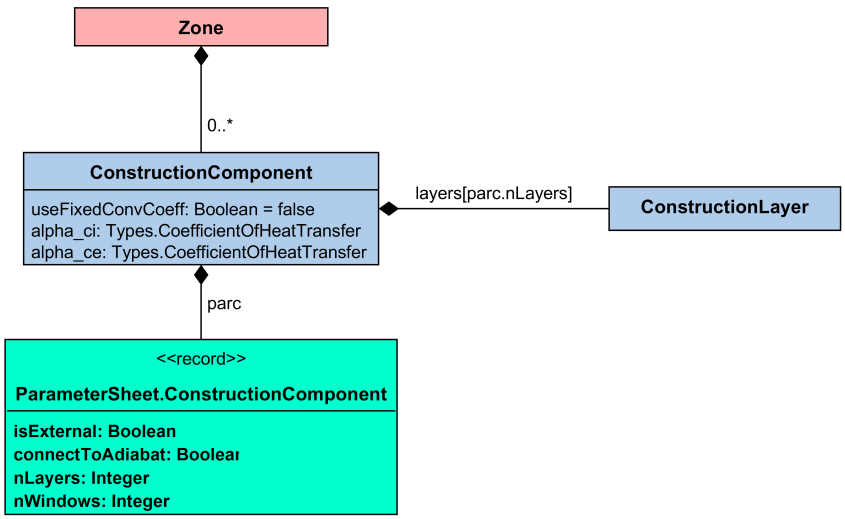


Abbildung 3.4: Beziehungen zwischen Zone, Bauelement und Materialschicht

3.3 Model View Definition: „Design to Building Energy Analysis“

Der Datenaustauschprozess zwischen Domänen in der Bauindustrie kann auch standardisiert werden. „Information Delivery Manual (IDM)“ ist ein von bSIveröffentlichter Standard, in dem eine Reihe von Prozessabbilder, Exchange Requirements Modelss (ERMs) und ein allgemeiner BIM-Leitfaden definiert werden. Hierbei wird es festgelegt, in welcher Planungsphase welche Daten von wem benötigt werden. Ein ERM besteht aus Diagrammen, die die benötigten Informationen – die benötigten Elemente und ihre Attribute in einem bestimmten Austauschprozess beinhalten. Model View Definitions (MVDs) sind IFC-spezifische Interpretationen von den ERMs. Sie beschreiben die Teilsätze von IFC-Datenschemen, damit nicht die ganze Datenmenge in einer IFC-Datei von Export zu Import übertragen werden soll, um die Effizienz für einen Datenaustausch zu bekommen und der Implementierung der Softwareschnittstellen zu helfen. (See *et al.*, 2012) Alle vorhandenen IFC-Dateien werden nach einem bestimmten MVD, die von den Softwareherstellern unterstützen, exportiert.

ERMs und MVDs sind als wiederverwendbare Werkzeuge zu betrachten. Viele öffentlich zugängliche MVD-Diagramme sind in der Entwicklung und werden veröffentlicht. Sie können auf den Webseiten http://www.buildingsmart-tech.org/specifications/ifc-view-definition und http://www.blis-project.org/IAI-MVD/MVDs herunterladen werden. Mit dem Referenznummer GSA-003 ist ein MVD mit dem Namen „Design to Building Energy Analysis (BEA)“ seit 2011 vorhanden. (See, 2011) Es beschreibt die grundlegenden Daten aus IFC von einem Architekturentwurf für eine Energiesimulation. Es wurde als ein Titel im National BIM Standard der USA aufgenommen und von den BIM-Plattformen wie Autodesk Revit, Graphisoft ArchiCAD, Dprofiler und DDS-CAD, sowie von den Simulationswerkzeugen wie EnergyPlus, RIUSKA und IDA ICE unterstützt. (National Institute of Building Sciences, 2012) Ein Übersichtsbild wird in Abbildung 3.5 gezeigt.

Im Mittelpunkt steht in diesem MVD die Raumdefinitionen, die ihre Geometrien, Identifikatoren und Raumbegrenzungen. Eine von bSI zertifizierte Software soll so implementiert werden, dass alle im MVD dargestellten Elemente zum Austausch bereitgestellt werden. Ein beispielhafter Ausschnitt von diesem MVD (Abbildung 3.6) zeigt die Darstellung. Darin sind die Elemente mit gestrichelten Rahmen optional zum Implementieren. Im Folgend wird es in den von Abschnitt 3.2 erwähnten Aspekten überprüft, wie MVD-BEA die von IBPSBuilding benötigten Daten umfassen kann.

3.3.1 Geometrie und Orientierung

Die Merkmale von Raumgeometrie werden im MVD-BEA als „Simple Element Quantities“ in „Space“ dargestellt. Darin sind Raumhöhe und Netto-Grundfläche mit eingeschlossen. Damit

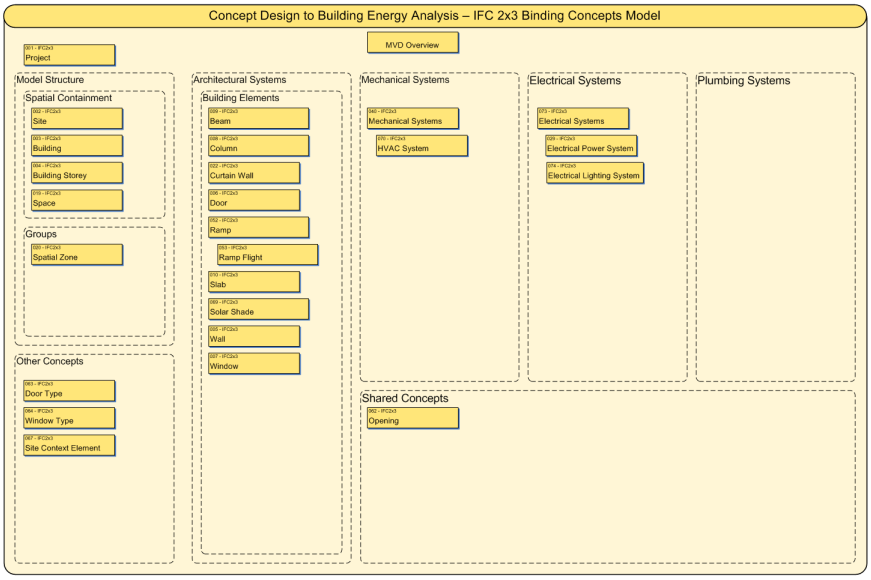


Abbildung 3.5: Übersicht des MVD-BEA

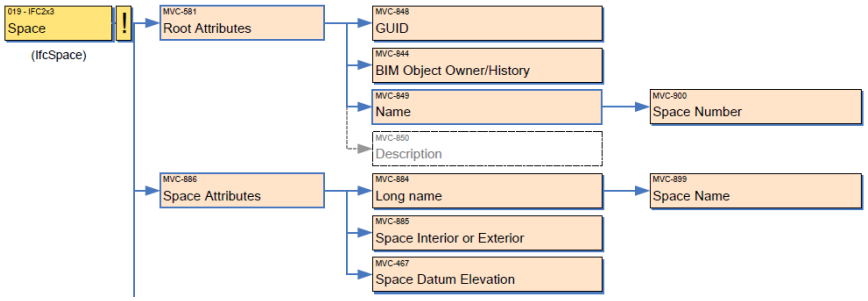


Abbildung 3.6: Ein Beispielhaftes Stück von MVD-BEA

werden diese Daten aus einer nach MVD-BEA exportierten IFC-Datei beim Export schon zusammengefasst. Ein zusätzliches Rechenverfahren ist dazu nicht erforderlich.

Im Vergleich dazu ist es komplizierter, die geometrischen Eigenschaften von Raumgrenzen von einer IFC-Datei auszulesen. Nach dem MVD-BEA werden nur Komponenten, die „Space Boundary Geometry“ beschreiben, exportiert. Sie werden durch Vektoren und Punkten in einem Koordinatensystem dargestellt und erfordern Berechnungen, um ihre charakteristischen Merkmale nach der Anforderung von einer Modelica-Energiesimulation (siehe Abschnitt 3.2) bereitzustellen.

Die Orientierung und die Neigung von einem Bauelement kommen aus Interpretationen der MVD-Komponenten „Local Relative Placement“. Räume, raumabschließende Oberflächen und Bauteilen haben ihre eigenen `IfcLocalPlacements`, um die Platzierungen zu beschreiben. Die Nordrichtung gilt als eine notwendige Hilfsinformation, um die Orientierung festzulegen. Sie wird im MVD-BEA als „3D Model Representation Types → True North“ in „Project“ dargestellt.

3.3.2 Materialeigenschaften

Materialeigenschaften sind wichtig für Energiesimulationen. Im IFC-Definitionsschema steht ein `IfcMaterialProperties`-Element, das durch das `Material`-Attribut mit einem `IfcMaterial` verbunden werden kann. Physikalische Daten mit einfachen Datentypen werden darin aufgelistet. In der 2X3 Version des IFCs werden Subtypen wie `IfcGeneralMaterialProperties` und `IfcThermalMaterialProperties` davon abgeleitet. Sie werden in der IFC4-Version in P-sets umgeschrieben, um den Implementierungsprozess zu vereinfachen. Tabelle 3.1 zeigt die Entsprechungen zwischen den benötigten Parametern für Materialien und den bezogenen IFC-Attributen.

Tabelle 3.1: Entsprechungen zwischen benötigten Materialeigenschaften und IFC-Attributen

Materialeigenschaften	IFC-Objekttyp[a]	Attribut
Massendichte	IfcGeneralMaterialProperties	MassDensity
Viskozität	**IfcExtendedMaterialProperties**	ViscosityTemperatureDerivative
Wärmespeicherfähigkeit	**IfcExtendedMaterialProperties**	SpecificHeatTemperatureDerivative
Wärmeleitfähigkeit	**IfcExtendedMaterialProperties**	ThermalConductivityTemperatureDerivative
Brechungsindex	**IfcExtendedMaterialProperties**	SolarRefractionIndex
Transmissionskoeffizient	IfcOpticalMaterialProperties	SolarTransmittance
Reflektionskoeffizient	IfcOpticalMaterialProperties	SolarReflectanceFront SolarReflectanceBack
Emissivität	IfcOpticalMaterialProperties	ThermalIrEmissivityFront ThermalIrEmissivityBack

[a] in der Version IFC2X3

Abbildung 3.7 steht für eine Teilmenge vom MVD-BEA, die die exportierten Materialeigenschaften aus IFC zu einer Energieanalyse darstellt.

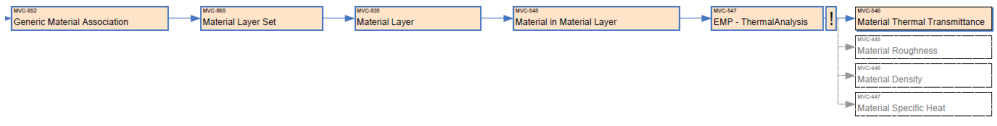


Abbildung 3.7: Materialeigenschaften in MVD-BEA

Das Element „Material in Material Layer“ ist eine Komponente vom Typ `IfcMaterial`, in dessen Definition der Identifikator, Kennung (ID) und der Name eines Materials aufgelistet werden. Die mit einem Material verbundenen Eigenschaften sind nur optional erforderlich: Hierzu kann nur das „IFC Extended Material Properties (EMP)“ laut des MVD-BEA exportiert werden. Im Vergleich zu Tabelle 3.1 sind die Informationen nur aus dem

`IfcExtendedMaterialProperties` nicht ausreichend. Überdies werden die für EMP erforderlichen Eigenschaften in Praxis auch nicht aus den meisten BIM-Softwareprogrammen exportiert. Daher entsteht eine Datenlücke für den Datenaustausch von exportierten IFC-Dateien zu Energiesimulationen.

Unvollständiger Informationsexport entsteht auch für Türen und Fenster. Nach MVD-BEA soll nur ein „Window Composite Thermal Transmittance“ exportiert werden. Dieser reicht für Simulationen nicht aus, da selbst für ein einfach modelliertes Fenster zumindest ein Gesamtenergiedurchlassgrad (g-Wert) benötigt wird.

3.3.3 Topologie

Die Topologie von einem simulierbares Modell in Modelica bedeutet Einbeziehungen und Verbindungen von den Bestandteilen miteinander. Ein MVD Add-Ons für Raumgrenzen werden von bSI angeboten, um aufzuzeigen, was auf der anderen Seite von einer begrenzenden Fläche eines Raums passiert. Nach Häfele (2009) kann eine Raumgrenze in IFC auf zwei Ebenen definiert werden. Eine Raumgrenze auf der 1. Ebene wird nur durch die Geometrie der Raum begrenzenden Oberfläche definiert, ohne zu berücksichtigen, ob ein Nebenraum oder ein dahinter stehendes Bauelement existiert. Im Gegensatz dazu ist die Erstellung der Raumgrenzen auf der 2. Ebene von den Randbedingungen abhängig. Die Raumgrenzen auf der 2. Ebene, die auf einem einheitlichen Bauteil basieren, aber auf der anderen Seite mit unterschiedlichen Räumen verbunden werden, werden als unterschiedliche Komponenten betrachtet. (siehe Abbildung 3.8) Für Energiesimulationen ist das Export der Raumgrenzen auf der 2. Ebene erforderlich. Das ist in MVD-BEA beschrieben. Zusätzliche Informationen fürs Festlegen der Randbedingungen eines Raums und einer Raumgrenze, wie das Attribut „Internal/External“, sind auch in diesem MVD mit enthalten.

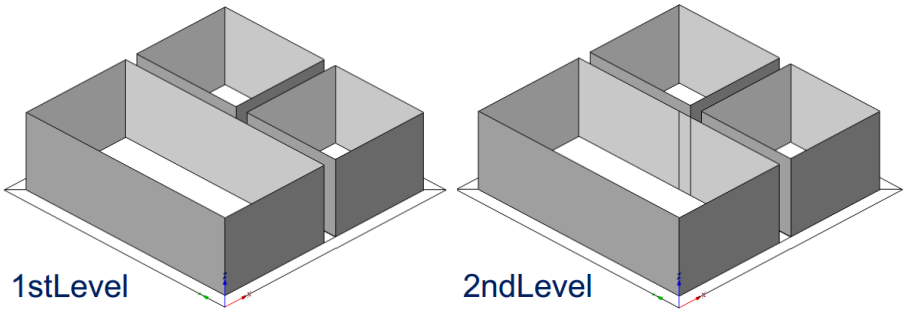


Abbildung 3.8: Unterschied zwischen der 1. Ebene und der 2. Ebene von der IFC-Definition einer Raumgrenze

Um die richtige Reihenfolge der Materialaufbauten festzulegen, sollte man sich auf die Definition, dass der Normalvektor von einer Raum begrenzende Oberfläche beim Export einer IFC-Datei immer nach außen zeigen, einigen. Das kann zurzeit beim Export-Prozess einer

IFC-Datei von den meisten in Praxis eingesetzten BIM-Softwareprogrammen nicht gewährleistet werden. (Maile *et al.*, 2013)

3.4 Zusammenfassung

Nachdem die Zieldaten (Modelica-Bibliothek) und Quelldaten (MVD-Beschreibung aus IFC) angegeben wurden, wird eine vollständige Vergleichstabelle (Tabelle C.1) erstellt. Hierzu wird gezeigt, welche Parameter mit welchen Definitionen aus dem MVD-BEA zugeordnet werden können. In der Spalte „Zustand“ wird dargestellt, ob die Daten für die Parameter aus IFC ausgelesen werden können. Man kann davon ausgehen, dass die meisten fehlenden Daten zu Material- und Fenstereigenschaften gehören. Obwohl die Komponenten dafür wie `IfcMaterialProperties` und `IfcDoorWindowGlazingType` in IFC-Definitionen vorhanden sind, aber nicht im MVD-BEA aufgelistet werden. Das heißt, dass sie zurzeit nicht von BIM-Software unterstützt werden. Eine Verbesserung und Erweiterung des MVDs wird deswegen verlangt.

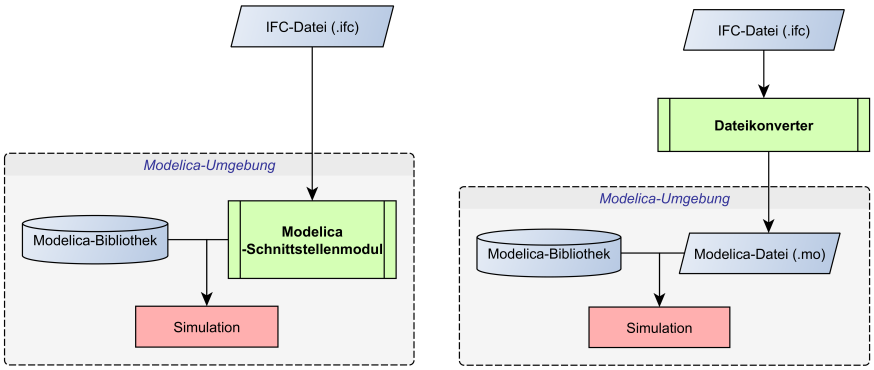
Kapitel 4

Implementierung einer Schnittstelle

In diesem Kapitel werden die Möglichkeit und die Methodik zur Implementierung einer Schnittstelle zwischen IFC und Modelica untersucht. Im Abschnitt 4.1 werden mögliche Aufbauten und ihre Funktionsprinzipien vorgestellt. Davon wird der Datenabrufprozess von einer der Methoden ausgewählt zu implementieren. Die in Java programmierten Komponenten in der sogenannte Energy Information EXchange (EIX)-Bibliothek werden im Abschnitt 4.2 eingeführt. Darüber hinaus wird ein Prototyp von einer einfachen Nutzeroberfläche für die Schnittstelle im Abschnitt 4.3 konzipiert, denn wegen der Vielfältigkeit von verfügbarer Modelica-Bibliotheken wird eine Flexibilität auf der Seite von den Nutzern eingefordert, damit die ausgetauschten Daten individuell angepasst werden können.

4.1 Methodik für die Implementierung und Aufbau der Schnittstelle

Unter dem Begriff Schnittstelle versteht man im weiteren Sinn verschiedene Formen von einer funktionellen Einheit, die den Austausch von Daten und Kommandos zwischen verschiedenen Systemen ermöglichen kann. Eine Schnittstelle für den Datenaustausch zwischen IFC und Modelica kann auf unterschiedlichen Ebenen von einer einfachen Codeergänzung bis zu einer vollständigen Anwendungssoftware aufgebaut werden. Abbildung 4.1 stellt zwei Möglichkeiten, wie die Schnittstelle im Austauschprozess eingesetzt werden kann. Abbildung 4.1a beschreibt entweder als ein integriertes Modelica-Modell oder eine Modelica-Funktion, die direkt in einer Modelica-Umgebung abgerufen werden kann. Im Vergleich dazu funktioniert sie als ein externer Dateikonverter wie Abbildung 4.1b gezeigt. In den folgenden Abschnitten werden die beiden Ansätze aufgelegt, damit es über ihre Ausführbarkeiten, Vorteilen, Schwierigkeiten und Beschränkungen diskutiert werden kann.



(a) integriertes Modelica-Modul **(b)** externer Dateikonverter

Abbildung 4.1: Unterschiedliche Einsatzebenen der Schnittstelle

4.1.1 Entwicklung einer Modelica integrierte Schnittstelle

In der Modelica-Sprache werden zwei Arten von Schnittstellen unterstützt: externe Funktionen und externe Objekte, bei denen extern gespeicherte C- und FORTRAN-Funktionen direkt in Modelica-Programmierung abgerufen werden können. (Association, 2012) Ein direkter Importprozess kann durch die Einsetzung der solchen Funktionen, die in z. B. `.c` und `.h` Dateien definiert werden, umgesetzt werden. Gäbe es eine externe Funktion für den Datenabruf eines bestimmten `ParameterSheet` (z. B. für eine Materialschicht), würde ein externes Objekt in Modelica wie folgend in Listing 4.1 geschrieben. Darin werden zwei extern in der C-Sprache definierten Funktionen, `void* retrieveIFCLayerParameter(const char*)` und `void closeLayerParameter(void*)` als Konstruktor (Zeile 4–8) und Destruktor (Zeile 10–13) verwendet. Eine zusätzliche `struct{}`-Struktur wird auch in der gleichen C-Datei definiert, um die `record`-Datenstruktur des Outputs abzubilden.

Listing 4.1: Aufbau eines externen Objekts

```
1  within ParameterSheet;
2  class Layer_ext
3      extends ExternalObject;
4      function constructor
5          input String ifcFileName;
6          output Layer_ext parl;
7          external "C" parl = retrieveIFCLayerParameter(ifcFileName);
8      end constructor;
9
10     function destructor
11         input Layer_ext parl;
12         external "C" closeLayerParameter(parl);
13     end destructor
14 end Layer_ext;
```

In Zukunft werden mehr Programmiersprachen von Modelica unterstützt. (Association, 2012) Mit der bisher vorgegebenen C-Schnittstelle ist es allerdings auch möglich, die Programmiersprachen, die direkte reguläre Verknüpfungen mit C besitzen, wie C++, und die Schnittstellen mit der C-Sprache anbieten, wie Java, in Modelica zu integrieren. Ein Beispiel wurde von Olsson (2005) dargestellt, dass ein Java-Dialog in Modelica-Simulation abgerufen werden kann.

Die Vorteile von einer Modelica integrierten Schnittstelle liegen im Vergleich zu einem Dateikonvertern klar auf der Hand: Beim Hinzufügen eines neuen Parameters wie `ifcFileName` und ein paar Gleichungen wie des Beispiels:

```
ParameterSheet.Layer_ext parl = ParameterSheet.Layer_ext(ifcFileName);
```

würde die Daten von der IFC-Datei einfach ins Modelica-Modell übertragen. Darüber hinaus steht dabei auch die Möglichkeit, während einer Simulation auf einem beliebigen Zeitpunkt die importierte IFC-Datei umzutauschen, oder durch die aktuellsten Simulationsergebnisse zu überschreiben. In der Folge könnte es damit eine Echtzeit-Entwurfsoptimierung ermöglichen.

Auf der anderen Seite hat eine integrierte Schnittstelle Nachteile. Davon auffällig ist es, dass die Flexibilität der Einpassung unterschiedlicher Modelica-Bibliotheken abgelenkt werden könnte. Die Modelle, welche die Daten aus IFC verwenden möchten, müssen nach der oben genannten Weise implementiert werden. Die Definitionen der externen `ParameterSheets` sollen immer gleichbleibend sein, um den C-Funktionen zu entsprechen. Außerdem ist es aufwändig, die Datenlücken aus einer unvollständigen IFC-Eingabe mit der Hand zu schließen, weil der Importprozess in Programmierung weit von den Nutzern abgeschottet wird.

Noch schlimmer stehen auch Beschränkungen in diesem Typ von Implementierung. Nur Parameterwerte können übertragen werden, aber keine Topologie von der IFC-Datei. Alle Verknüpfungen, inkl. die zwischen verschiedenen Bauteilen und Räumen, sollen in der Modelica-Bibliothek vorprogrammiert werden und unverändert bleiben. Deswegen ist diese Implementierungsmethode geeigneter für den Datenabruf aus einer Datenbank, die z. B. in Extensible Markup Language (XML) kodiert wird, wie von Tiller (2005) vorgestellt. Außerdem wird ein Array als ein Rückgabewert von einer externen Funktion bisher in Modelica (Version 3.3) nicht erlaubt. Das macht die Erstellung einer Array enthaltenden Parameterliste (z. B. Normalvektoren) praktisch unmöglich.

4.1.2 Entwicklung eines externen Dateikonverters

Ein anderer Weg von Implementierung besteht darin, dass ein externer Dateikonverter vom IFC-Dateiformat zu einem Modelica-Modell mit dem Suffix „`.mo`“ eingerichtet werden kann. Im Vergleich zu einer Modelica integrierten Schnittstelle funktioniert ein Dateikonverter wie ein unabhängiges komplettes Anwendungsprogramm, das nach Bedarf an verschiedene Modelica-Bibliotheken angepasst werden kann. Zusätzliche Nutzeroberflächen (Abschnitt 4.3)

sollen dafür aufgebaut werden, um die Flexibilität auf der Nutzer-Seite zu erreichen. Die Schnittstelle ist nicht bei Modelica-Codes abrufbar, sondern kann von Nutzern direkt in der Systemumgebung ausgeführt werden. Die Modelica-Bibliotheken wechseln hierzu ihre Rolle: Sie sind nicht mehr Anwender der Schnittstelle wie der in Abschnitt 4.1.1 erläuterte Funktionsablauf, sondern dienen als Inputs neben der IFC-Datei, und bieten Modellstrukturen in der Generierung eines simulierbaren Modelica-Modells an. Das Funktionsprinzip wird in Abbildung 4.2 dargestellt.

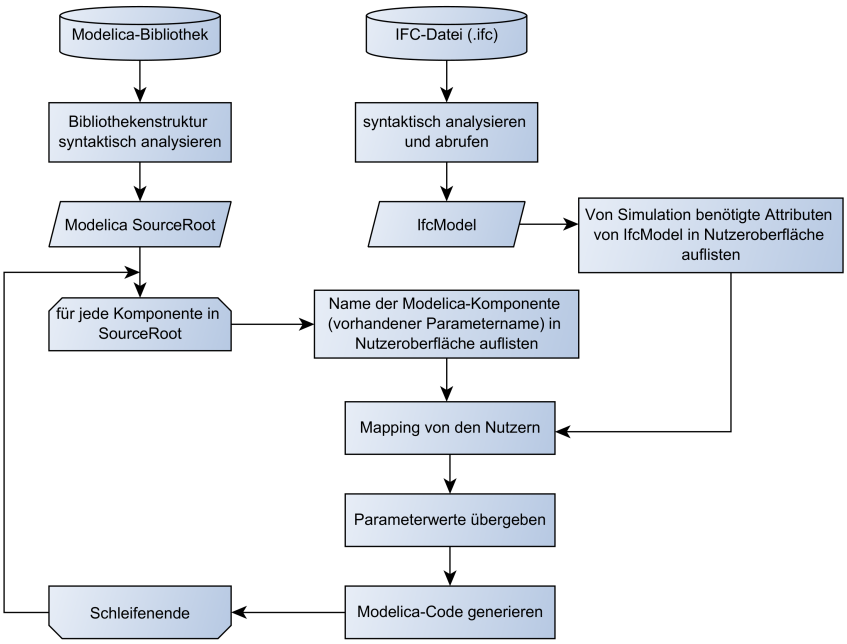


Abbildung 4.2: Arbeitsablauf eines Konverters

Im folgend stellt eine Übersicht dar, um die Begriffe von den in der Abbildung gezeigten Prozessen (in den rechteckigen Rahmen) zu erklären und Querweise innerhalb dieses Artikels jeweils anzubieten:

Syntaxanalyse einer IFC-Datei

Für die Implementierung beim Rechnern reicht das Verständnis und die Darstellung der IFC-Dateistruktur wie im Abschnitt 3.1 nicht aus. Die Implementierung der Schnittstelle fängt mit einem Analyseprozess an, damit eine IFC-Datei syntaktisch in ihre grammatikalischen Bestandteile (od. Token) zerlegt werden kann. Damit können die IFC-Codezeilen in strukturierte Programmierobjekte umgewandelt werden. In Abschnitt 4.1.3 wird es darüber ausführlich erzählt.

Syntaxanalyse einer Modelica-Gebäudebibliothek

In gleicher Weise soll die Modelica-Gebäudebibliothek, auf der das generierte simulierbare Gebäudemodell basiert, auch syntaktisch analysiert werden. Ziel besteht es darin, dass die Struktur des Packages vom Rechnern erkannt werden kann. Sie wird in der Modelica-Code-Generierung umgesetzt. Dafür sind Parser aus Toolkits wie JModelica verfügbar. Die Modellstrukturen und ihre Variablen werden nach der Zerlegung vom `ModelicaCompiler` aus JModelica in einem `SourceRoot`-Modell als Java-Klassen umgewandelt. Für jede Komponente mit dem Präfix `parameter` innerhalb des `SourceRoot`-Modells soll durch eine Reihe von entwickelten Methoden mit den Daten, die aus dem IFC abgerufen werden, zugeordnet werden.

von Simulation benötigte Attribute aus IFC abrufen

Die Daten, die in dem zerlegten `IfcModel` gespeichert werden, werden nach dem Bedarf, der in Kapitel 3 erläutert wurde, durchgesiebt. Eine Klassenbibliothek mit Klassendefinitionen, die von den IFC-Komponenten aufgebaut werden (deren Konstrukturargumente von den IFC-Komponenten übergeben werden), kann damit programmiert werden. Eine beispielhafte Implementierung – EIX wird in Abschnitt 4.2 systematisch vorgestellt. Die Implementierung von einer Energiesimulation-spezifischen Modellbibliothek, die von den IFC-Klassendefinitionen erweitert wird, erhöht dabei die Wiederverwendbarkeit der Codes, damit nicht nur ein Typ von Code-Generierungsprozess, sondern egal die Entwicklung von einer integrierten, externen oder Schnittstelle als Middleware die Codes wie EIX nützen kann.

Modelica-Komponenten, Mappings mit Attributen aus IFC und Modelica-Codegenerierung

Parameter aus den beiden syntaktisch zerlegten Modellen werden in einer Nutzeroberfläche aufgezeigt. Ein konzeptioneller Entwurf von der Graphical User Interface (GUI) wird in Abschnitt 4.3 dargestellt. Aktivitäten wie Übergeben der Parameterwerte werden von den Nutzern durch die GUI erledigt.

Das komplette Konverter-Programm kann als ein Bestandteil in ein Middleware eingefügt werden, das auf einer Seite die IFC-Dateien zentral verwalten und verarbeiten kann und auf der anderen Seite ein Modelica-Compiler (z. B. OMC in OpenModelica-Toolbox) abrufen kann, um Modelica-Simulationen für die IFC-dargestellten Gebäude laufen zu lassen. Somit kann das Middleware als eine BIM-Plattform fungieren. Der Konverter wäre auch dafür geeignet, dass er als ein Plugin in eine vorhandene Modelica-Simulationsumgebung integriert werden kann. Abbildung 4.3 stellt einen Ausblick von einem Anwendungsszenerio dar, dass der Konverter in einem Middleware eingesetzt wird.

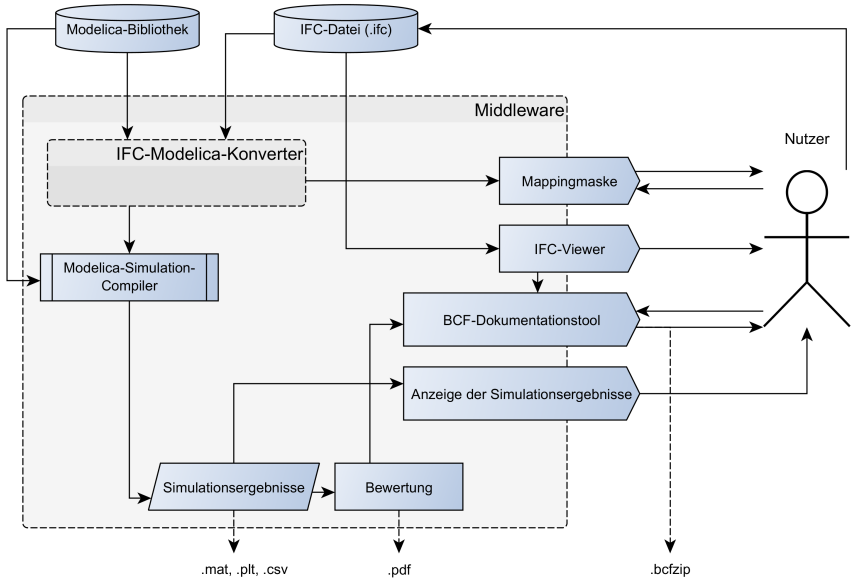


Abbildung 4.3: Konverter als ein Bestandteil in einem konzeptionellen BIM-Middleware

4.1.3 Syntaxanalyse einer IFC-Datei

Der Begriff einer Syntaxanalyse wurde in Abschnitt 4.1.2 erklärt. Dieser Analyseprozess ist allerdings nicht nur für die Entwicklung einer externen Dateikonverter erforderlich, sondern auch bei der Implementierung einer integrierten Schnittstelle. Vielmehr ist es notwendig, vor allen auf IFC bezogenen Programmentwicklungen die Syntaxanalyse auszuführen. Nur syntaktisch zerlegte Modelle können in einer Programmiersprache verarbeitet werden und mit eigenen programmierten Komponenten zusammenarbeiten. Zahlreiche Open-Source-Toolkits wurden entwickelt, um die Syntaxanalyse (od. Parsing) durchzuführen. Diese Werkzeugsätze wurden in bestimmten Sprachen entwickelt, oder in bestimmten Entwicklungsumgebungen integriert, damit die zerlegten Token direkt weiterverarbeitet werden können.

Das im Abschnitt 3.1 genannte „IFC TOOLS Project“ (Tauscher, 2013) bietet außer den Modell-Viewers auch ein Java-Development-Kit (JDK) (auch als IFC-Java-Toolbox bekannt) an, das beim Abrufen, Erstellen und Modifizieren IFC-Dateien in einer Java-Entwicklungsumgebung helfen kann. In der aktuellsten Version kann nicht nur IFC2x3, sondern auch das neue IFC4 verarbeitet werden. Das Toolbox beinhaltet eine Reihe von Java-Klassen, welche die in den EXPRESS-Schemen definierten IFC-Entitäten repräsentieren, und ein Parser (dt. Zerteiler), das IFC-STEP-Dateien und IFCZIP-Dateien einlesen kann. Die beiden wird von einem Zentralmodell mit dem Namen `IfcModel` verwendet, das bei eigener Programmierung wie Listing 4.2 instanziiert werden kann, damit Daten in IFC-Dateien abgerufen werden können. Spezifikationen stehen auf der Webseite http://www.ifctoolsproject.com/ zur Verfügung. (Theiler, 2013)

Listing 4.2: mit *IfcModel* in IFC-Java-Toolbox eine IFC-Datei abrufen (Theiler, 2013)

```
1 //create a new instance of IfcModel
2 IfcModel ifcModel = new IfcModel();
3
4 //load an IFC STEP file from the file system
5 File stepFile = new File("C:\\myfile.ifc");
6 ifcModel.readStepFile(stepFile);
```

In ähnlicher Weise gibt es ein anderes Open-Source-Werkzeug „IfcPlusPlus“ (http://www.ifcplusplus.com/), in dem die IFC-Klassen und Parser in C++ geschrieben werden. Es könnte dazu mehr Coding-Aufwand im Einsatz der Weiterentwicklung fordern, aber erhöht die Effizienz des CPUs und Arbeitsspeichers. Außerdem bietet der Werkzeugsatz ein IFC-Geometrie-Konverter an, der die Geometrie-Komponenten in einheitliche Fläche-Kante-Eckpunkte-Darstellungen umwandeln kann. (Google Project Hosting, 2014) Ein robuster Einsatz, welcher der Microsoft Windows .NET-Plattform entspricht, heißt eXtensible Building Information Modelling (xBIM)-Toolkit und wird auf der Webseite http://xbim.codeplex.com/ vorgestellt. Es besteht aus 8 C#-Projekte und 1 C++-Projekt, in denen C#-Klassen, Parser, Viewer, eine Unterstützung für Construction Operations Building Information Exchange (COBie) und Konvertierung der Geometrie umfasst werden. (OpenBIM, 2014)

In dieser Masterarbeit wird das Java-Toolbox aus „IFC TOOLS Project“ verwendet. Alle Quellcodes werden in Java in der Umgebung „Eclipse“ geschrieben.

4.2 EIX-Bibliothek

In Abschnitt 4.1.2 wurde der Begriff von der Energy Information EXchange (EIX)-Bibliothek vorgestellt. In diesem Abschnitt wird ihrer Aufbau beschrieben. Das Ziel liegt darin, der Abrufprozess der benötigten Daten aus IFC-Dateien in Programmiersprache zu interpretieren. Es steht auch im Mittelpunkt des Arbeitsbereiches, für das die Baufachleute in der Implementierung eines Datenaustauschprozesses verantwortlich sind. Die verarbeitete Bibliothek kann für verschiedene Weiterentwicklungsansätze von Informatikern zur Verfügung gestellt werden.

Ein Übersichtsbild, das in Abbildung 4.4 dargestellt wird, zeigt die Grundstruktur der Bibliothek „`eixComponents`“ mit zurzeit implementierten Komponenten. Der Inhalt untergliedert sich nach der Logik, die in Kapitel 3 eingeführt wurde, in drei Aspekten, die die auszutauschenden Informationen umfassen. Hierzu zählen Platzierung und Geometrie, Materialeigenschaften und Verbindungen zwischen Elementen.

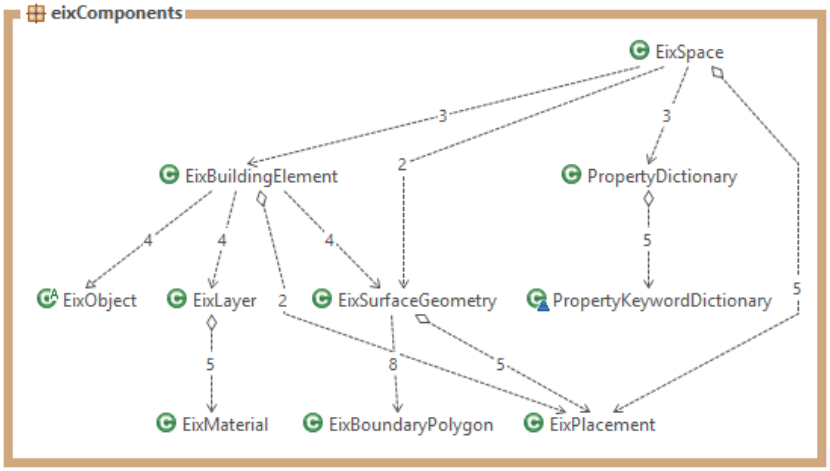


Abbildung 4.4: Übersicht des Zustands der EIX-Bibliothek

4.2.1 Geometrie und Platzierung der Räumen und Bauteilen

Raumgeometrie

Räume stehen im Mittelpunkt in einer Energiesimulation. (2.3) In der EIX-Bibliothek wurde eine `EixSpace`-Komponente dafür implementiert, in der fünf grundlegenden Attribute definiert wurden. Dazu zählt eine Nettogrundfläche, ein Volumen, eine spezifische Raumhöhe, eine `List` von Raumgrenzen im Typ `EixBuildingElement` und eine Lage im Typ `EixPlacement`, der später in diesem Artikel erzählt wird. Als das einzige Argument für ihren Konstruktor ist ein `IfcSpace` beansprucht. In Abschnitt 3.3 wurde es erläutert, dass nach der Anforderung vom MVD-BEA die Basismengen exportiert werden sollen. Sie sind im Typ `IfcElementQuantity` (ein Subtyp von `IfcRelDefineByProperties`) in der IFC-Datei gespeichert, und durch das Attribut `IsDefinedBy_Inverse` innerhalb eines `IfcSpace`-Modells mit den Raumdaten verknüpft werden. Diese geometrischen Mengen werden durch andere „`set...`“-Funktionen aus dem `IfcModel` herausgegriffen. Die Funktionen werden vom Konstruktor abgefragt, damit der komplexe IFC-Verarbeitungsprozess schlicht verkapselt werden kann. Das komplette Klassendiagramm von `EixSpace` wird in Abbildung 4.5 aufgezeigt.

Schwierigkeiten können von der Anwendung dieser `IfcElementQuantity`-Komponenten mitgebracht werden. Diese `IfcElementQuantity`-Elemente haben einen gleichen Datentyp und unterscheiden sich nur durch die Namen. Die exportierten Namen sind abhängig von der Software, bei dem die IFC-Datei erstellt wird. Die Software könnte in einer englischen, deutschen, oder anderssprachigen Version verfasst werden. Um die EIX-Komponenten die Elementnamen richtig verstehen lassen zu können, wurde ein Wörterbuch `PropertyKeywordDictionary` und eine kleine Suchfunktion – `PropertyDictionary.findWord(String, String)` entwi-

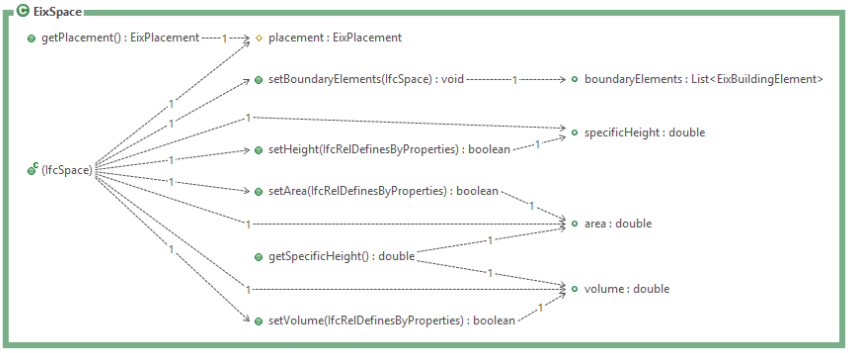


Abbildung 4.5: Klassendiagramm des *EixSpace*

ckelt. Das Wörterbuch sieht wie Listing 4.3 aus und kann durch neue Wörter und neue Sprachen erweiter werden.

Listing 4.3: Beispielhaftes Wörterbuch für IfcProperties

```
protected PropertyKeywordDictionary(){
    this.put("net", new HashSet<String>(Arrays.asList("net","netto")));
    this.put("area", new HashSet<String>(Arrays.asList("area","flaeche")));
    this.put("volume", new HashSet<String>(Arrays.asList("volume","volumen"))
        );
    this.put("height", new HashSet<String>(Arrays.asList("height","hoehe")));
    this.put("length", new HashSet<String>(Arrays.asList("length","laenge")))
        ;
}
```

Für die IFC-Dateien, die keine Basismengen wie Raumhöhe und Grundfläche erhalten, wird die Raumgeometrie durch Abfragen des Wertes im Typ `IfcProductDefinitionShape` vom IFC-Attribut `Representation` ausgerechnet. Der Zugriffspfad ist ähnlich wie der für die geometrischen Daten von einer Raumgrenze, die im nächsten Abschnitt erwähnt wird, aber er wird zurzeit in der EIX-Bibliothek erst nicht implementiert, weil diese unvollständigen IFC-Dateien die Anforderung von MVD-BEA, die in Abschnitt 3.3 dargestellt werden, nicht erfüllen.

Geometrie der raumabschließenden Bauteilen

Anders als IFC haben die Definitionen der von Raum unabhängigen Bauteilen in EIX-Bibliothek keine Bedeutung, weil sie in den thermischen Prozessen vernachlässigt werden sollen, und daher die Simulationsverfahren nicht beeinflussen. In jeder `EixSpace`-Komponente wird ein `List` von Raumgrenzen beinhaltet, die im Typ `EixBuildingElement` stehen. In ihrer Definition werden die Eigenschaften aus einem `IfcBuildingElement`-

Modell (oder einem Modell in seinem Subtyp, z. B. `IfcWall` oder `IfcSlab`) und die Eigenschaften der Raumgrenzen, die durch das `BoundedBy`-Attribut im `IfcSpace` beschrieben werden, zusammengefasst. Dabei soll eine `IfcBuildingElement` nur dann abgefragt werden, wenn das `BoundedBy` von einer Raumkomponente sie verlangt. In der Funktion `setBoundaryElements(IfcSpace)` in der Klasse `EixSpace` wird es gezeigt, dass der Abruf von raumabschließenden Bauteilen sich nur auf das betrachtete `IfcSpace` bezieht. Der Konstruktor von einer `EixBuildingElement`-Komponente erfordert zwei Argumente: ein `IfcBuildingElement` und ein `EixSurfaceGeometry`, das von einem `IfcConnectionSurfaceGeometry` umgewandelt wird. Abbildung 4.6 stellt den Abrufsablauf der beiden benötigten Argumente aus einer `IfcSpace`-Komponente dar. Das erzählt auch das Grundprinzip, nach dem die Funktion `setBoundaryElements(IfcSpace)` implementiert wurde.

Abbildung 4.7 stellt die Grundstruktur der `EixBuildingElement`-Klasse dar. Die Parameterwerte für Geometrie, `perimeter` und `area` werden vom `EixSurfaceGeometry`, das als ein Argument von dem Konstruktor eingesetzt wird, festgelegt. Das heißt, die geometrischen Eigenschaften von den raumabschließenden Bauteilen kommen nur aus `IfcConnectionSurfaceGeometry`-Komponenten, und sind unabhängig von den geometrischen Repräsentationen der Instanzen der Bauelementen wie `IfcWall` oder `IfcSlab`. Dadurch ist der Repräsentationstyp eines Bauelements, `Extrusion` oder `BRep`, keiner Bedeutung für die Erstellung einer Raumgrenze.

Um das Rechenverfahren für die geometrischen Daten innerhalb eines `EixSurfaceGeometry` zu erzählen, wird erst die Klasse `EixBoundaryPolygon` vorgestellt. Die Klasse enthält nichts aber nur ein `List` von Eckpunkten vom Typ `Point3d`[1], die die Koordinatenwerte von den Eckpunkten eines einfachen `IfcPolyline` übertragen. Die Reihe der `Point3d`-Eckpunkte stehen für die Randlinie einer Oberfläche. Die Funktion, die die Fläche und den Umfang ausrechnet, wurde nach Formeln 4.1 und 4.2 implementiert. (Page, 2014)

$$A = \sum_{i=0}^{n} (x_i + x_{i+1})(y_i - y_{i+1}) \tag{4.1}$$

$$P = \sum_{i=0}^{n} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{4.2}$$

wobei $x_{n+1} = x_0$, $y_{n+1} = y_0$.

Zwei `List`-Elemente vom Typ `EixBoundaryPolygon` werden in der Definition `EixSurfaceGeometry` beinhaltet, um die Geometrien von äußeren und inneren Randlinien (siehe Abschnitt 3.3) ei-

[1] Um den Typ *Point3d* zu benutzen, wird das Package *javax.vecmath* erst in die Programmierung eingeführt. Dieses Package und seine API-Dokumente können auf der Webseite https://java3d.java.net/ herunterladen werden. Darin sind Definitonen für Punkten, Vektoren, Matrizen und die Rechenoperationen dafür mit eingeschlossen.

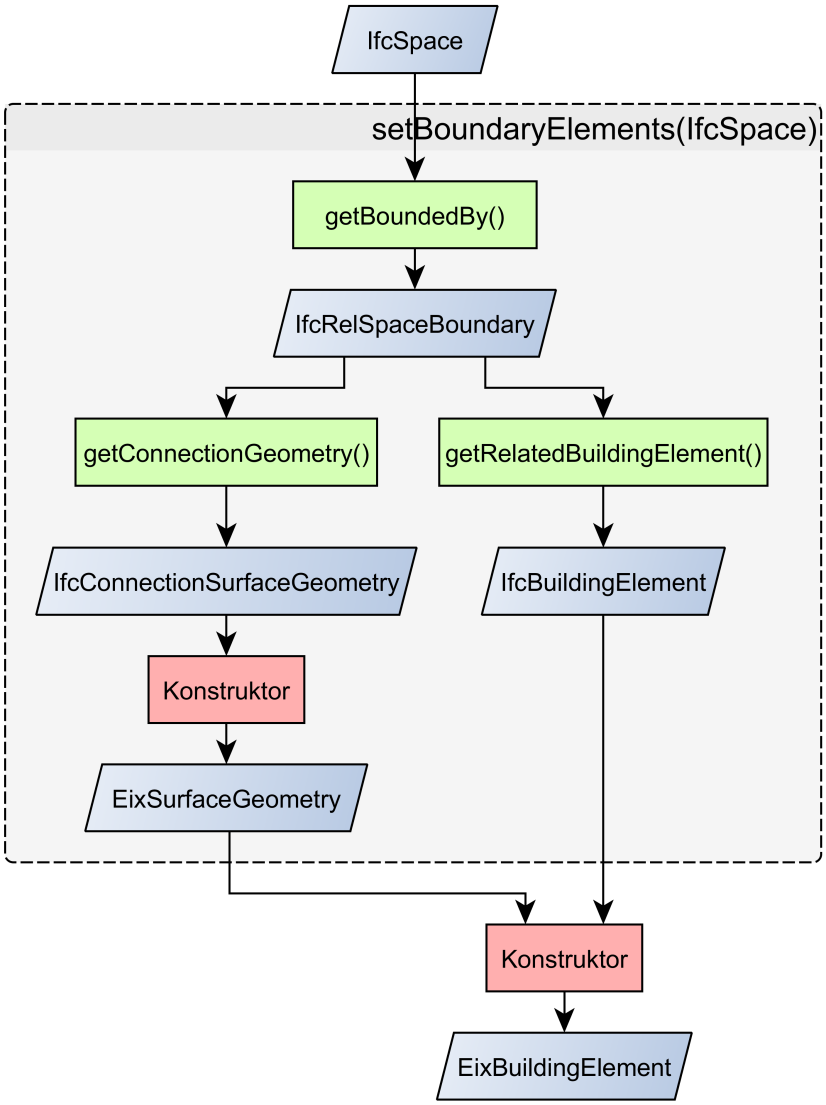


Abbildung 4.6: Flussdiagramm für den Abrufsprozess der bezogenen Informationen von den Raumgrenzen aus $IfcSpace$

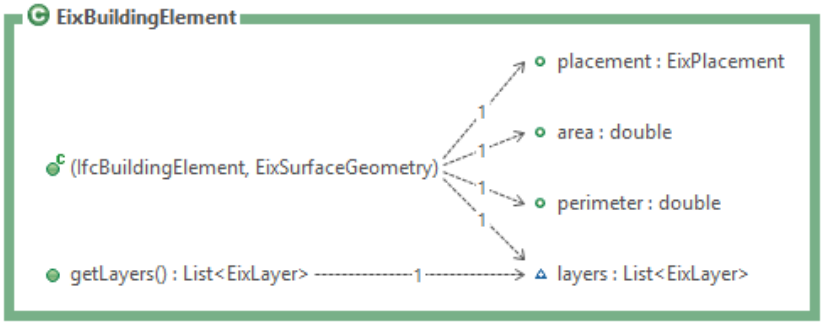


Abbildung 4.7: Klassendiagramm des *EixBuildingElement*

ner Oberfläche aufzunehmen. Die Nettofläche einer Raumgrenze ist die Differenz zwischen die gesamten Flächen der beiden. Ein Flussdiagramm in Abbildung 4.8 stellt dar, wie die vom Konstruktor des `EixBoundaryPolygon` benötigten `IfcPolylines` bei der Erstellung eines `EixSurfaceGeometry` abgefragt werden.

Platzierung (Lage)

In der Definitionen `EixSpace`, `EixBuildingElement` und `EixSurfaceGeometry` sind Komponenten `EixPlacement` als Mitglieder vorgegeben. Sie zeichnen jeweils die Lagen der Objekte auf, wo die Objekte in einem globalen Koordinatensystem platziert werden. Ein `EixPlacement` besteht aus einem Ursprungspunkt (`location`) und drei normalisierten orthogonalen Vektoren (`vecX`, `vecY` und `vecZ`), welche die drei Achsen repräsentieren. In einer `EixSurfaceGeometry`-Komponente enthaltenen Randlinien aus `IfcPolyline` werden durch 2D-Punkten beschrieben, die sich allen auf der xy-Ebene setzen. Deswegen gilt die Z-Achse des `EixPlacement` als der Normalvektor der Oberfläche, die diese Lagenkomponente einschließt. Abbildung 4.9 zeigt den Grundaufbau der `EixPlacement`-Klasse.

Die beiden Funktionen, `getTiltAngle(Eixplacement)` und `getAzimuth(Eixplacement)`, bieten die Möglichkeit, die Neigungswinkel und Azimuthwinkel (Orientierung) anhand der Koordinatenwerte des Ursprungs und der Achsenvektoren auszurechnen. Das benötigte Argument, `Eixplacement global`, steht für das in der IFC-Datei vordefinierte globale Koordinatensystem und die Nordrichtung[2]. Wenn eine manuelle Eingabe für die Nordrichtung nicht existiert, wird die umgekehrte Richtung der X-Achse als vorgegeben angewandt. Das heißt, dass für das vorgegebene IFC-Globalsystem $((1,0,0),(0,1,0),(0,0,1))$ die Nordrichtung als $(-1,0,0)$ definiert wird (siehe Abbildung 4.10). Damit entspricht sie der gewöhnlichen Regel, wobei Nord–0°, Ost–90°, Süd–180°, West–(−90°) ist. Das IBPSBuilding-Package in Modelica verwendet auch diese Regel, um die Orientierung festzulegen.

[2]Die Nordrichtung kann extern per Hand eingegeben werden. Das kann bei Interaktionen durch die Nutzeroberfläche (Abschnitt 4.3 in Zukunft umgesetzt werden.)

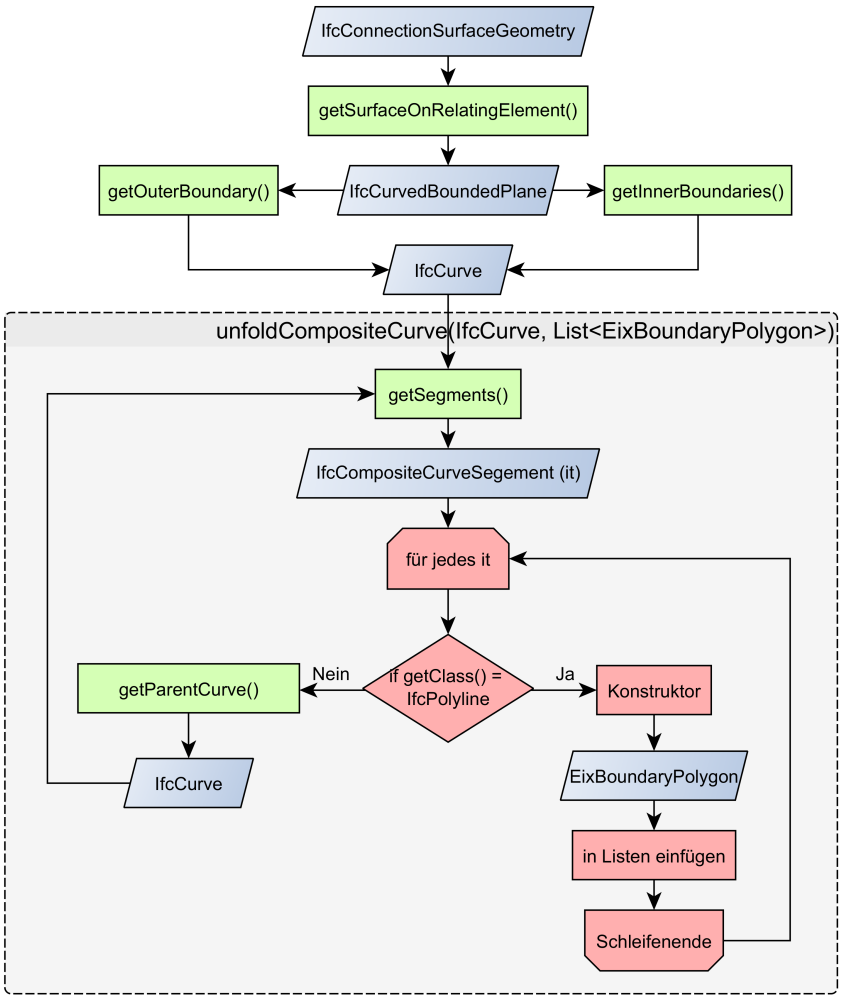


Abbildung 4.8: Flussdiagramm für den Abruf $IfcPolylines$ von $IfcConnectionSurfaceGeometry$

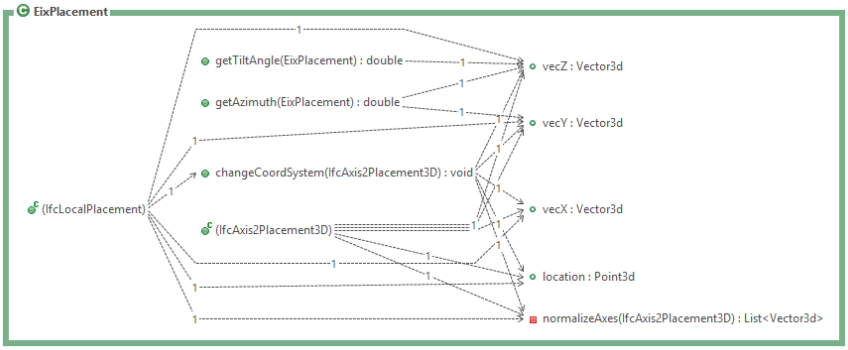


Abbildung 4.9: Klassendiagramm des $EixPlacement$

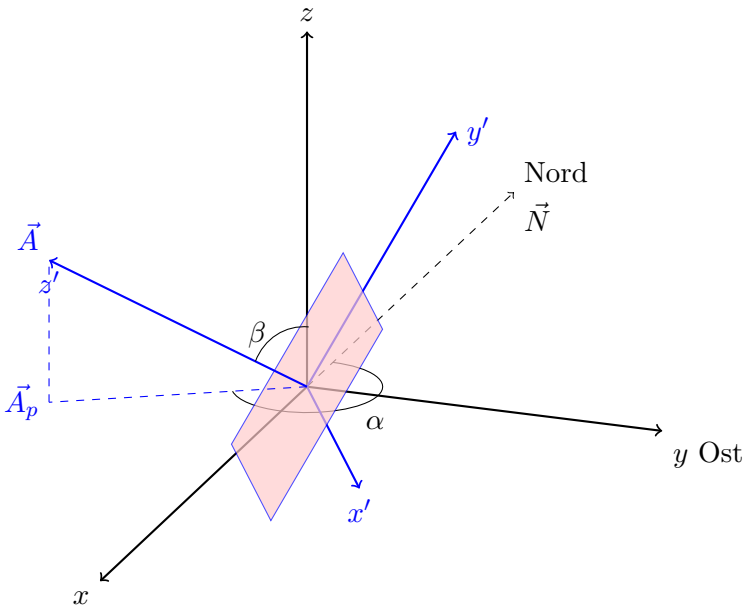


Abbildung 4.10: Festlegen des Neigungs- und Azimuthwinkels nach dem Globalsystem und der Nordrichtung

In Abbildung 4.10 wird das Grundprinzip der Implementierung der beiden Funktionen dargestellt. Das schwarze xyz-Koordinatensystem steht für das Globalsystem, und das blaue $x'y'z'$ repräsentiert das Lokale, das in jedem `EixPlacement` abgespeichert wird. Dabei für die betrachtete Oberfläche-Komponente hat der Azimuthwinkel α, der Neigungswinkel β, ihr Normalvektor $\vec{A}$ und seine Projektion $\vec{A}_p$ die folgenden Zusammenhänge, die in Formel 4.3 und Formel 4.4 dargestellt werden:

$$\beta = \arccos\left(\frac{\vec{A} \bullet \vec{Z}}{\left\|\vec{A}\right\| \left\|\vec{Z}\right\|}\right) \tag{4.3}$$

$$\vec{A}_p = \vec{A} - \left(\vec{A} \bullet \vec{Z}\right) \cdot \vec{Z},$$
$$\alpha = \begin{cases} \arccos\left(\frac{\vec{A_p} \bullet \vec{N}}{\left\|\vec{A_p}\right\| \left\|\vec{N}\right\|}\right) & \text{falls } \vec{A}_p \bullet \vec{Y} >= 0 \\ -\arccos\left(\frac{\vec{A_p} \bullet \vec{N}}{\left\|\vec{A_p}\right\| \left\|\vec{N}\right\|}\right) & \text{falls } \vec{A}_p \bullet \vec{Y} < 0 \end{cases} \tag{4.4}$$

Nach der Erzählung von den Anwendungen der `EixPlacement`-Klasse (Erfassen der charakterischen Daten), soll der Erstellungsprozess einer `EixPlacement`-Komponente im folgend beschrieben werden. Zwei Konstruktoren wurden dafür implementiert: Einer davon baut die Komponente aus einem `IfcAxis2Placement3D` auf. Im Vergleich dazu benutzt der Andere ein

`IfcLocalPlacement` als das Argument. Der letztere zielt darauf, ein lokales Koordinatensystem, das auf einer überlagerten Ebene tief versteckt, direkt dadurch ins „Root“-Globalsystem umgewandelt werden kann. Ein Funktion mit dem Namen „`changeCoordSystem`“ wird innerhalb des Konstruktors verwendet, um die Transformation des Koordinatensystems umzusetzen. Für das in einem oberen Koordinatensystem T $((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$ eingeschlossene Subsystem T' $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ wird ein Rechenverfahren (4.5) eingeführt (University Oslo, 2003).

$$\mathbf{a} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{4.5}$$

Gleichfalls können $\mathbf{b}$ und $\mathbf{c}$ aufgelöst werden.

Ein Übersichtsbild wird in Abbildung 4.11 dargestellt, um den Abwicklungsprozess der lokalen Systeme ins Globale zu beschreiben.

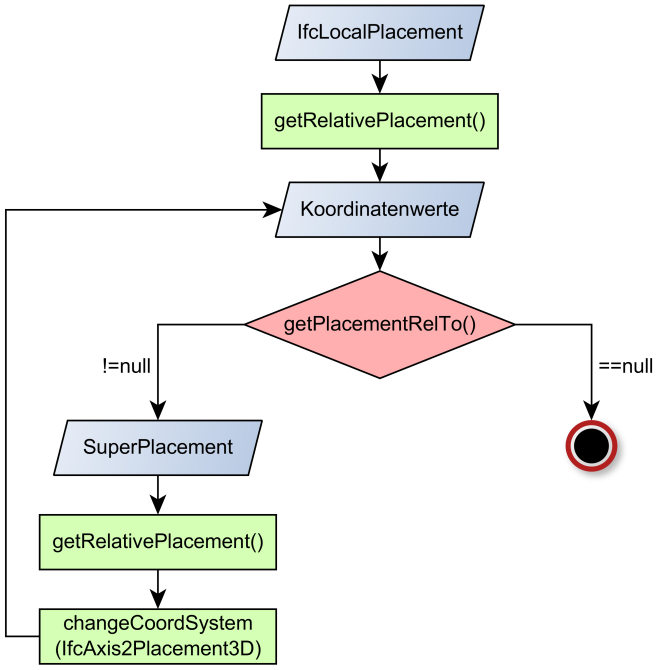


Abbildung 4.11: Flussdiagramm des Konstruktors $EixPlacement(IfcLocalPlacement)$

Jedes `IfcCurveBoundedPlane`, das innerhalb eines `IfcConnectionSurfaceGeometry` durch die `getSurfaceOnRelatingElement()`-Methode gebunden wird, besitzt eine Subkomponent im Typ `IfcLocalPlacement`, die die Lage der Fläche aufzeichnet, und bei `getPosition()` abgefragt werden kann. Es wird als das Argument im Konstruktor eines `EixSurfaceGeometry` zugeordnet, um sein `placement`-Attribut festzulegen. Die Lage einer Oberfläche wird ebenfalls zum `EixBuildingComponent`, zu dem die Oberfläche gehört, übermittelt. Nach diesem Pro-

zess können die Lageinformationen für jeder Raumgrenze abgerufen werden. Durch den Abruf der Funktionen `getTiltAngle(EixPlacement)` und `getAzimuth(EixPlacement)` können die Neigungen und Orientierungen, die von Energiesimulationen beansprucht werden, ausgerechnet werden.

4.2.2 Materialeingenschaften

In Abschnitt 3.3.2 wurde die Datenlücke von IFC2x3 und seine Abgrenzungen im Bereich Materialeigenschaften erläutert. Ein nahtloser Datenaustauschprozess zwischen IFC und Modelica-Energiesimulationen wird daher behindert, weil die in der aktuellsten MVD fehlenden Materialeigenschaften (Wärmeleitfähigkeit, Wärmespeicherfähigkeit und Massendichte) fürs Simulieren des Wärmeübertragungsprozesses notwendig sind. Als eine Zwischenlösung kann die Schnittstelle den Nutzern die Möglichkeit anbieten, dass sie durch die GUI entsprechende brauchbare Materiallisten aus der Modelica-Gebäudebibliothek (z. B. IBPSBuilding-Package) für die IFC-Materialien entscheiden können. Deswegen soll eine Methode trotz der Unvollständigkeit einer IFC-Datei für den Abruf Materialien implementiert werden.

Die `EixMaterial`-Klasse wurde einfach wie in Abbildung 4.12 aufgebaut. Der Namenwert eines `EixMaterial` wird durch die Funktion `IfcMaterial.getName()` zugeordnet. Er fungiert als ein Index, mit dem ein passender Materialtyp aus der Modelica-Bibliothek von Nutzern ausgewählt werden kann. Die Materialkomponenten, die die gleichen Namen haben, gelten theoretisch als eine einheitliche Art Material, aber ihre Attributwerte können individuell nach Situation angepasst werden.

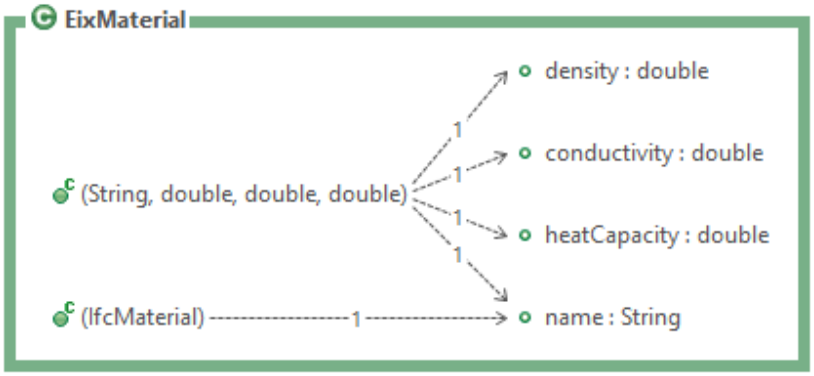


Abbildung 4.12: Klassendiagramm des *EixMaterial*

Ein `EixLayer` ist eine Komponente, die ein `EixMaterial` besitzt und in deren Definition die Dicke der Materialschicht aufgezeichnet wird. Eine Reihe von `EixLayer` wird innerhalb eines `EixBuildingElement` angeordnet, damit der Schichtaufbau eines raumabschließenden Bauteils dargestellt wird. Abbildung 4.13 zeigt, wie die Materialschichten im Konstruktor eines `EixBuildingElement` festgelegt werden.

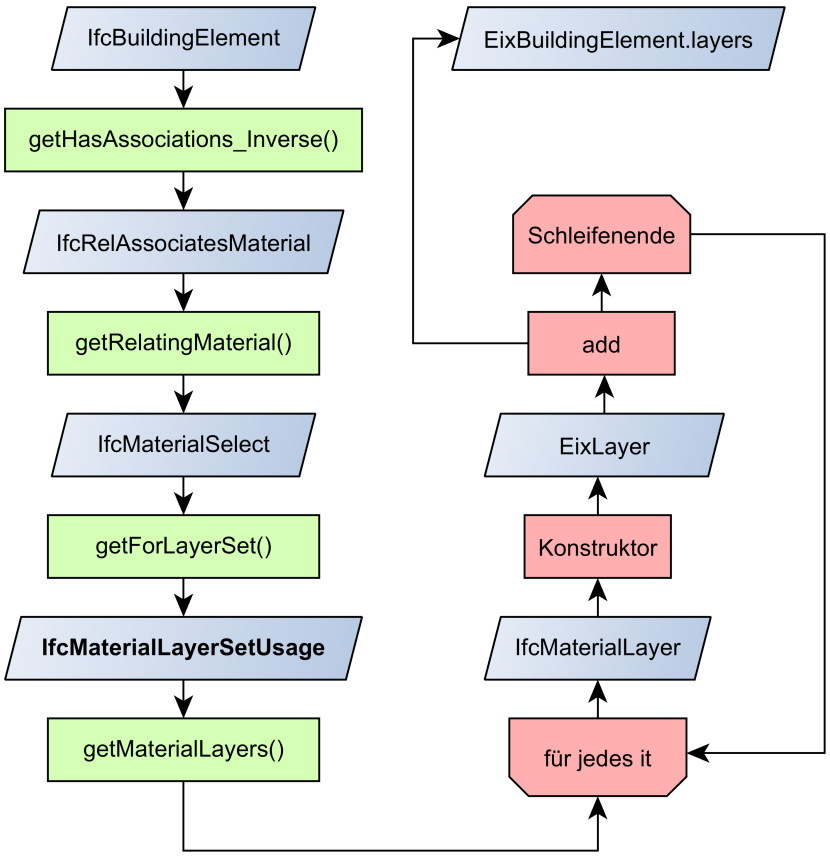


Abbildung 4.13: Flussdiagramm für die Erstellung *EixLayer*-Komponenten im Konstruktor eines *EixBuildingElement*

Die Reihenfolge von Schichtaufbauten eines `EixBuildingElement` wird so geregelt, dass das erste Element des Arrays der Materialschichten innerhalb jedes `EixBuildingElement` mit der innerste Materialschicht, die nach dem Raum richtet, betrachtet wird. Damit werden die in Modelica importierten Bauteilen bei der „`connect`“-Gleichungen, die durch die Codegenerierungsphase entstehen, in die richtigen Richtungen der Wärmetransmission im Raum eingelegt und mit dem Luftknoten verknüpft. Abbildung 4.14 zeigt, wie die richtige Reihenfolgen der Materialschichten (Array `EixLayer` innerhalb eines `EixBuildingElement`) festgelegt werden kann. In der Definition von `IfcMaterialLayerSet`, das durch das Abrufen von Attributwerten mit einem `IfcBuildingElement` verknüpft wird, wird der Vektor, entlang deren Richtung die Materialschichten aufgebaut werden, durch einem Attribut mit dem Namen `LayerSetDirection` erstellt. Für ein beliebiges `EixBuildingElement` wird es so umgesetzt: Wenn die Normalvektor, der durch die Platzierung im Attribut abgespeichert wird, die gleiche Richtung mit der vom Attribut `LayerSetDirection`, das vom in seinem Konstruktor als Argument geltenden `IfcBuildingElement` beinhaltet wird, besitzt, wird das Array-Attribut `EixLayer` innerhalb dieses `EixBuildingElement` nach der gleichen Reihenfolge wie die Schichten im `IfcBuildingElement` zugeordnet. (Siehe Raum1) Auf der anderen Seite wie in Ab-

bildung 4.14 „Raum2“ gezeigt, wenn die zwei Normalrichtungen vom `EixBuildingElement` und `IfcMaterialLayerSet.LayerSetDirection` unterschiedlich sind, soll die Aufbaurichtung der Reihe von Materialschichten umgekehrt sein. Das heißt, dass das erste Element vom `EixLayer` von der letzten Schicht in diesem `IfcBuildingElement` zugeordnet werden soll.

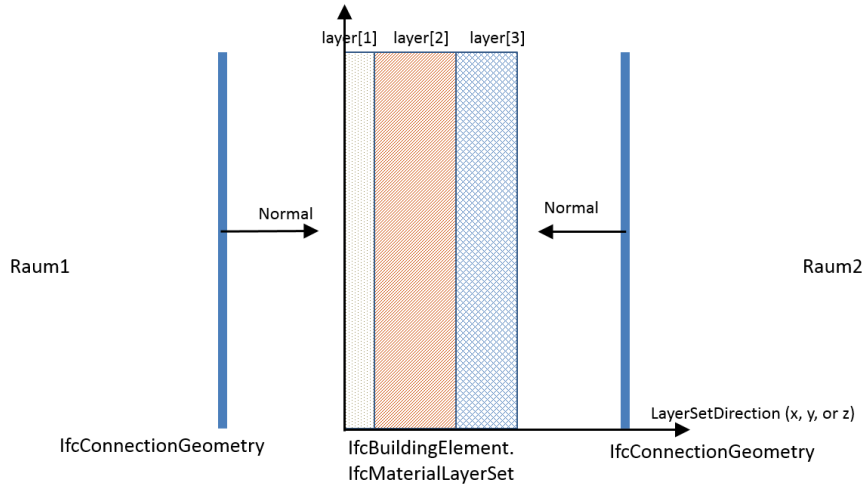


Abbildung 4.14: Festlegen der Reihenfolge vom Aufbau der Materialschichten eines *EixBuildingElement*

4.2.3 Topologie

Betrachtet man die Topologie, die von IFC zu einer EIX-Struktur interpretiert werden soll, um für die Generierung eines Modelica-Diagramms (2.3) bereitgestellt werden zu können, werden die folgenden drei wesentlichen Aspekten berücksichtigt.

Randbedingungen

Eine Reihe von `EixBuildingElement`-Komponenten sind in der Definition von `EixSpace` mit eingeschlossen. Jede Komponente hat ein Attribut mit dem Namen `bCondition` im Aufzählungstyp `BoundaryCondition`, in dem drei Werte: `EXTERNAL`, `INTERNAL` und `ADIABAT` aufgelistet werden. Die Zuordnungsmethoden werden dafür noch nicht vollständig implementiert. Das Attribut `InternalOrExternalBoundary` vom Typ `IfcRelSpaceBoundary`, mit dem die Raum begrenzende Oberfläche und das Bauteil miteinander verknüpft werden, kann dazu als Quellinformation benutzt werden. Das `bCondition`-Attribut bietet ein Anhaltspunkt für Generierung der „`connection()`“-Sätze in einem Modelica-Raummodell, damit es erklärt werden kann, an welchen Anschlüssen (äußeren, inneren, mit Nebenräumen oder mit einer fixen Temperatur) die Raumgrenze angeschlossen werden sollen.

Nebenraum und gemeinsam genutzte Bauteile

Ein `EixBuildingElement` ist abhängig nur von einem Raum. Das heißt, dass es nur zu einem Raum gehören soll. Fürs Festlegen von zwei Nebenräumen, die durch eine gemeinsame Trennwand aufgeteilt werden, sollen die nebeneinander stehenden `EixBuildingElement`-Komponenten von den zwei Räumen, die ein gleiches physikalisches Bauteil darstellen, und in deren Konstruktoren das gleiche `IfcBuildingElement` eingesetzt werden, erst festgelegt werden. Die Festlegungsmethode untergliedert sich in zwei Schritte:

1. Die GUIDs von jeden zwei `EixBuildingElements`, die von den im Konstruktor beschriebenen `IfcBuildingElements` abgeleitet wurden, werden miteinander verglichen. Die Komponenten, die einen gleichen IFC-GUID besitzen, verweisen das gleiche Bauteil auf den unterschiedlichen Seiten von den zwei Nebenräumen.

2. Betrachtet man das Szenario, dass eine Grenzoberfläche von einem Raum wegen der unterschiedlichen Randbedingungen auf der anderen Seiten (z. B. mit mehreren Räumen verbunden) in mehrere `IfcConnectionSurfaceGeometries` zerteilt werden könnte (siehe Abschnitt 3.3.3 und Abbildung 3.8), soll ein weiterer Vergleich umgesetzt werden, um die genaueren Kopplungen von den nebeneinander stehenden Raumgrenzen festzulegen. Dazu sollen alle entsprechenden geometrischen Eigenschaften von den beliebigen zwei `EixBuildingElement`-Komponenten, die die gleichen GUIDs haben und zu unterschiedlichen Räumen gehören, verglichen werden. Wenn alle geometrischen Daten aus einem Paar davon gleich sind, gelten die zwei den Raum begrenzenden Bauteilen als ein gemeinsam genutztes Bauteil.

Die zwei Seiten eines gemeinsam genutzten Bauteils (od. die zwei `EixBuildingElments`, die zu zwei Räumen gehören und nach der ober genannten Methode herausgefunden werden), sollen durch eine neu generierte Modelica-„`connect()`“-Gleichung verbunden werden. Die hier erläuterte Methode wird aber erst nicht im Stand der EIX-Bibliothek abgespeichert, sondern diese kann bei der Weiterentwicklung für Modelica-Codegenerierung beschrieben werden.

4.2.4 Test für den Datenabruf aus IFC

Ein Testmodell wurde programmiert, um die implementierten Teile von den Datenabrufprozesse aus IFC mittels der `EIX`-Bibliothek zu validieren. Hierzu wurde ein 2-Zonen-Modell in Revit aufgebaut, und als eine IFC2x3-Datei mit Raumgrenzen auf 2. Ebene (siehe Abschnitt 3.3.3 und Abbildung 3.8) exportiert. Der Grundriss des Modells wird in Revit wie in Abbildung 4.15 dargestellt.

Im Test wurde nur die grundlegenden geometrischen Daten für Räume und ihre Raumgrenzen, die aus der IFC-Datei zusammengefasst werden, ergeben. Hierzu zählen unter anderem:

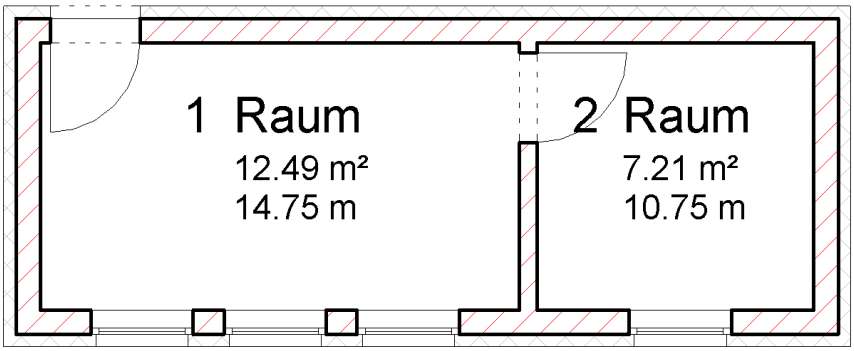


Abbildung 4.15: Das in Revit gebaute 2-Zonen-Modell für den Test des Datenabrufs

der IFC-Liniennummer, die Fläche, das Volumen und die Netto-Raumhöhe der Räumen, die Fläche und der Umfang der Raumgrenzen, sowie ihre Materialien. Die Reihenfolgen der Materialschichten wurden nicht nach der in Abschnitt 4.2.3 erwähnten Methode angepasst.

Die Ausgabe wird im Konsole aufgezeigt (siehe Abbildung 4.16). Nach dem Vergleich zwischen diesen Werten und den Eigenschaftswerten direkt aus Revit kommt es zu der Schlussfolgerung, dass die geometrischen Merkmale von diesem IFC-Modell richtig erfolgreich mittels `EIX` abgerufen werden können.

```
Room 129 has a location at (0.0, 0.0, 0.0)
        the area of its boundary surfaces:
        Its floor area = 12.49, volume = 34.98, height = 2.97
                boundary 1: 1023
                        area: 11.74; perimeter: 15.93
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
                boundary 2: 904
                        area: 0.13; perimeter: 6.09
                        material: Mauerwerk - Ziegel, verputzt,
                boundary 3: 5919
                        area: 1.03; perimeter: 4.06
                        material:
                boundary 4: 704
                        area: 12.74; perimeter: 15.93
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
                boundary 5: 6018
                        area: 1.03; perimeter: 4.06
                        material:
                boundary 6: 904
                        area: 6.68; perimeter: 11.84
                        material: Mauerwerk - Ziegel, verputzt,
                boundary 7: 9288
                        area: 2.09; perimeter: 6.49
                        material:
                boundary 8: 6083
                        area: 1.03; perimeter: 4.06
                        material:
                boundary 9: 4471
                        area: 2.09; perimeter: 6.49
                        material:
                boundary 10: 4582
                        area: 15.00; perimeter: 16.00
                        material: Stahlbeton - Ortbeton,
                boundary 11: 1711
                        area: 8.90; perimeter: 11.93
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
```

(a) Raum 1

```
Room 437 has a location at (0.0, 0.0, 0.0)
        the area of its boundary surfaces:
        Its floor area = 7.21, volume = 20.20, height = 2.93
                boundary 1: 4582
                        area: 9.00; perimeter: 12.00
                        material: Stahlbeton - Ortbeton,
                boundary 2: 6264
                        area: 8.76; perimeter: 11.84
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
                boundary 3: 9364
                        area: 1.03; perimeter: 4.06
                        material:
                boundary 4: 6350
                        area: 7.73; perimeter: 11.84
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
                boundary 5: 9288
                        area: 2.09; perimeter: 6.49
                        material:
                boundary 6: 704
                        area: 0.24; perimeter: 5.77
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
                boundary 7: 904
                        area: 6.68; perimeter: 11.84
                        material: Mauerwerk - Ziegel, verputzt,
                boundary 8: 6176
                        area: 8.52; perimeter: 11.84
                        material: Wärmedämmung - Hart, Mauerwerk - Ziegel, verputzt,
```

(b) Raum 2

Abbildung 4.16: Testausgabe für den Datenabrufprozess aus dem Konsole

4.3 Entwurf einer einfachen Nutzeroberfläche

In diesem Abschnitt wird eine Skizze von einer möglichen Nutzeroberfläche(GUI) eines externen IFC-Modelica-Dateikonverters, der nach der in Abbildung 4.2 gezeigten Struktur aufgebaut wird und die EIX-Komponenten verwendet, dargestellt. Wie in Abbildung 4.17 gezeigt

werden die in der abgerufenen IFC-Datei verwendeten EIX-Definitionen und ihre Attribute rechts aufgelistet. Links sind die zuzuordnenden Parameter aus der ausgewählten Modelica-Bibliothek. Durch die „Drag-and-Drop“ Aktion, ein Element von der `EixComponents`-Liste in ein entsprechendes Feld in der Modelica-Tabelle zu verschieben, wird die Kopplung der beiden Elementen zugewiesen. Es ist möglich, die fehlenden Informationen per Hand in die leere Felder eingegeben werden zu können.

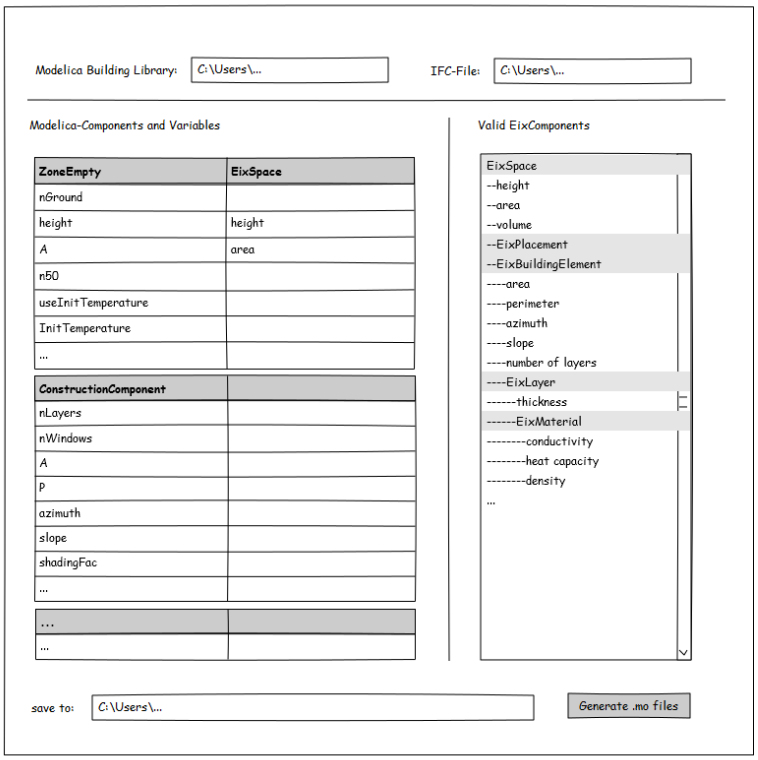


Abbildung 4.17: Skizze der GUI für den IFC-Modelica-Dateikonverter

4.4 Zusammenfassung

Die in diesem Kapitel dargestellten zwei Methoden für die Implementierung der Schnittstelle haben beide Vorteile und Nachteile, sodass auf keine in der zukünftigen Weiterentwicklung verzichtet werden kann. Der Stand der Implementierung, der mit dieser schriftlichen Arbeit auf einer CD gespeichert wird, zielt auf die Implementierung eines externen Dateikonverters. Der Entwurf der Nutzeroberfläche und den damit verbundenen Funktionen wurde nur konzeptionell vorgestellt, aber nicht in Codes programmiert. Die EIX-Bibliothek gilt als eine Zusammenfassung von der in Energiesimulationen verfügbaren Daten, die aus IFC abgerufen werden können. Die Attribute und Methoden können nicht nur für eine Entwicklung

eines Dateikonverters, sondern auch für eine Entwicklung der in Modelica-Codes integrierten Schnittstellen verwendet werden. Dazu wird die Unterstützung von mehreren objektorientierten Programmiersprachen in Zukunft erwartet.

Nicht alle zugreifbaren Daten nach Tabelle C.1 können bei der zurzeit implementierten EIX-Methoden abgerufen werden. Methoden für die Informationen über Topologie wie Randbedingungen, Anzahl der Materialschichten und Anzahl der Fenster werden nicht implementiert. Sie sollen erst nach der Validierung und Vervollständigung des IBPSBuilding-Packages eingefügt werden.

Weitere Unvollständigkeiten sind in der Vielfältigkeit der IFC-Dateien begründet. Geometrien, die in einer IFC-Datei nicht durch `IfcCurveBoundedPlane` (für Oberflächen) und —`IfcCurve` (für Randlinien) sondern mit anderen Definitionen beschrieben werden, werden bei der Erstellung der EIX-Komponenten übersehen oder Ausnahmemeldungen und unerwartete Fehler können bei der Durchführung des Programms ausgegeben werden.

Im nächsten Schritt wird die EIX-Bibliothek so erweitert, dass sie für die neuen Version IFC4 geeignet ist. Die Parser sind schon in der Java-Toolbox vorhanden. Schnittstellen für Materialeigenschaften und Fenster-Komponenten sollten in Zukunft ergänzt werden.

Kapitel 5

Ausblick

Entwicklung einer Schnittstelle für Modelica-Energiesimulationen gilt heutzutage als ein neues heißes Forschungsthema. Projekte und Lehrveranstaltungen darüber laufen parallel bei vielen Forschungseinrichtungen. Eine Forschungsgruppe von der Texas A&M University hat Schnittstellen dafür entwickelt, die durch die Application Programming Interface (API) von Revit die Revit-Modelle direkt in Modelica-Modelle für Energiesimulationen umwandeln können. Als die Ziel-Bibliothek wird „Modelica Buildings Library“ hierzu verwendet. (Yan *et al.*, 2013) Im Vergleich dazu sind die in dieser Masterarbeit erläuterten Implementierungsmethoden unabhängiger von Anwendungsprogrammen und Plattformen.

Basierend auf den in Modelica vordefinierten externen Funktionen werden verfügbare Schnittstellen in vielen Modelica-Umgebungen angeboten. Um sie zu vereinheitlichen, wird ein Konzept – Functional Mock-up Interface (FMI) eingeführt. Dieses gilt als ein Standard zwischen verschiedenen Simulationswerkzeugen, damit der Informationsaustausch und die Simulationskopplung ermöglicht wird. (Olsson, 2005) Die meisten Modelica-Umgebungen und bekannte Simulationssoftwareprogrammen wie TRNSYS unterstützen FMIs. Es ist in Zukunft möglich, ein Simulationssystem in einer auf Gebäude konzentrierten Software wie TRNSYS, in denen Schnittstellen zu BIM-Modellen vorhanden sind, aufzubauen und das durch FMI als ein Kopplungsmodul in ein Modelica-Modell zu integrieren, damit es während der Simulationszeit mit anderen in Modelica geschrieben Bestandteilen zusammenwirken kann. Im Vergleich dazu stellen die Methoden in dieser Masterarbeit eine aufwändigere Arbeit dar, aber bieten reinere Lösungen nur in Modelica-Sprache, um eine höhere Leistung und eine Verbesserung der Nutzerfreundlichkeit zu gewährleisten.

Mit der entwickelten Schnittstelle kann ein Modelica-Simulationsprozess mit der Arbeitsweise von BIM eng verbunden werden. Nicht nur IFC-Dateien für Gebäude sondern andere STEP-Dateien, die Produkte in anderen Fachbereichen wie TGA beschreiben, können in einem System in der Modelica-Umgebung integriert werden. Damit kann das „Building Information Modeling“ zum „Product Information Modeling and Simulation“ erweitert werden. In Zu-

kunft ist es zu erwarten, dass ein bidirektionaler Datenaustausch zwischen BIM-Modellen und Modelica-Simulationen eingesetzt werden kann. Das heißt, dass nicht nur die Gebäudedaten aus IFC, wie in dieser Arbeit erwähnt, in eine Modelica-Simulation importiert werden können, sondern auch eine Datenübertragung in der umgekehrten Richtung ausgeführt werden kann: Die Simulationsergebnisse können während der Simulation in die IFC-Datei eingeschrieben werden, damit sie auch zentral abgespeichert werden können, um für alle Projektbeteiligten zur Verfügung zu stehen.

Modelica-Systeme sind modular aufgebaut, und sind für Erweiterungen sehr geeignet. Um mit BIM zusammenzuwirken, wird eine standardisierte Beschreibung für Gebäuden auf unterschiedlichen Detaillierungsgraden erforderlich. Die Standardisierung verlangt offene Quellen der verschiedenen Beteiligten. Verwendbare Produkte für Schnittstellen dazwischen zu entwickeln, ist keine einfache Arbeit. Deshalb ist die Förderung von Open-Source-Projekten in diesem Bereich besonders wichtig.

Anhang A

Für den Datenaustausch benötigte Modelle von IBPSBuilding-Package

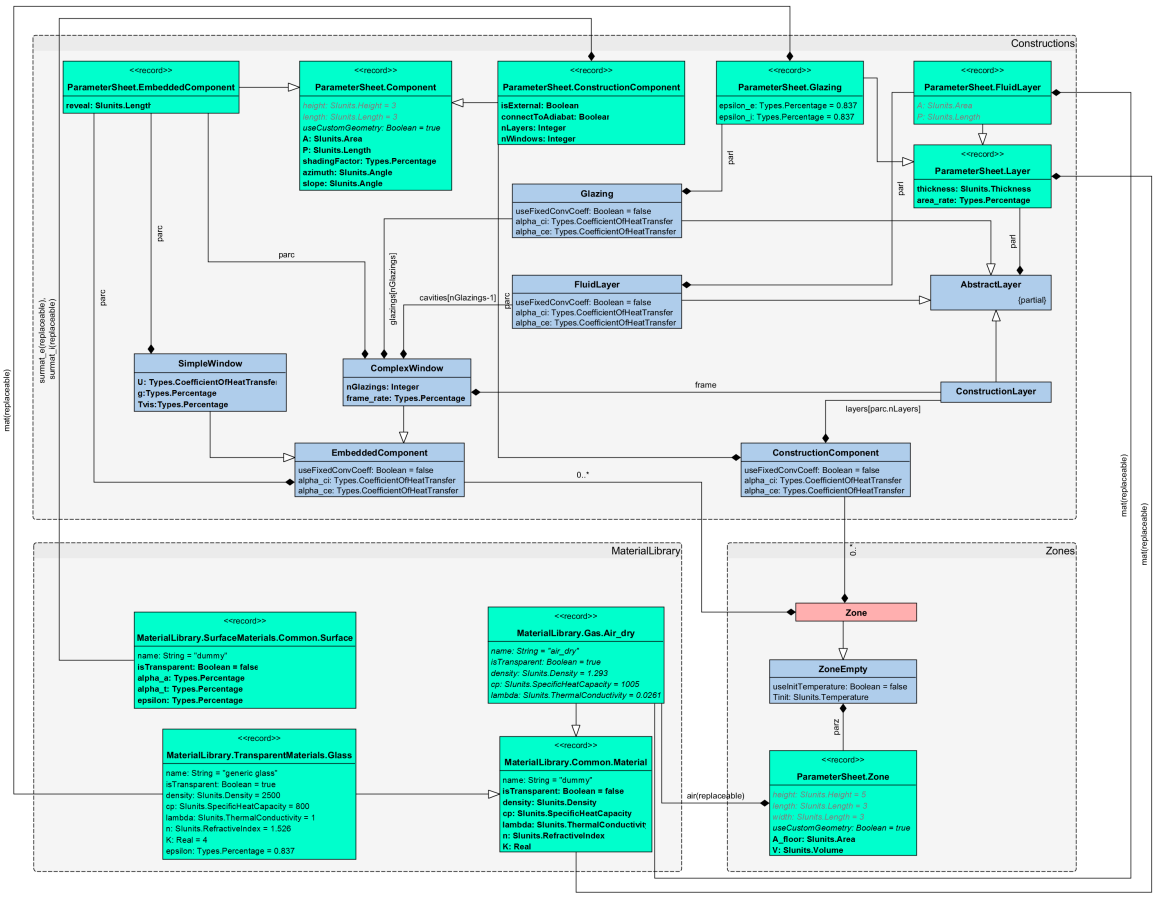


Abbildung A.1: Übersicht aller von IBPSBuilding-Package benötigten Modelle und Parameter

Anhang B

Kopplungen zwischen EIX- und IFC-Komponenten

Abbildung B.1: Kopplungen zwischen EIX- und IFC-Komponenten

Anhang C

Zuordnung der IBPS-Parameter mit Datenschemen aus MVD-BEA

Tabelle C.1: Zuordnung der IBPS-Parameter mit Datenschemen aus MVD-BEA

IBPS	MVD-BEA	Zustand	Weiterentwicklung
Zone			
A_floor	ElementQuantities	voll zugreifbar (v.z.)[1]	
V	ElementQuantities	v.z.	
Bauteil			
A	SpaceBoundaryGeometry	v.z.	
P	SpaceBoundaryGeometry	v.z.	
shadingFactor	for Window: DoorWindowShadingType	fehlt[2]	
azimuth	SpaceBoundaryPlacement	v.z.	
slope	SpaceBoundaryPlacement	v.z.	
isExternal	SpaceBoundaryInternalorExternal	v.z.	
connectToAdiabat	SpaceBoundary	beschränkt zugreifbar (b.z.)[3]	

Fortsetzung auf nächster Seite

[1] „Voll zugreifbar“ bedeutet, dass die Information für diesen Parameter nach der Beschreibung vom MVD aus IFC ohne Problem ausgegriffen werden kann

[2] „Fehlt“ bedeutet, dass die Information in den Definitionen innerhalb der IFC-Datenschemen fehlt.

[3] „Beschränkt zugreifbar“ bedeutet, dass die Information implizit im MVD enthalten wird.

Tabelle C.1 – Fortsetzung von vorheriger Seite

IBPS	MVD-BEA	Zustand	Weiterentwicklung
nLayers	count SpaceBoundaryRelatedBuildingElement	v.z.	
nWindows	count SpaceBoundaryRelatedBuildingElement	v.z.	
Window			
reveal	–	b.z.	
U	Window Property	in IFC[4]	DoorWindowGlazingType
g	–	in IFC	DoorWindowGlazingType
nGlazings	–	in IFC	DoorWindowGlazingType
frame_rate	GlazingAreaFraction	b.z.	IFC4: WindowFrameProperties
Materialschicht			
thickness	Layer thickness	v.z.	
area_rate	–	fehlt	IFC4: MaterialConstituent
density	–	in IFC	MaterialPropertiesEnergy/Thermal
cp	–	in IFC	MaterialPropertiesEnergy/Thermal
lambda	–	in IFC	MaterialPropertiesEnergy/Thermal
n	–	in IFC	MaterialPropertiesEnergy/Thermal
k	–	fehlt	
alpha_a	–	in IFC	MaterialPropertiesOptical
alpha_t	–	in IFC	MaterialPropertiesOptical
epsilon	–	in IFC	MaterialPropertiesOptical

[4] „In IFC“ bedeutet, dass die bezogene Information in IFC-Datenschemen steht, aber nicht im MVD dargestellt wird. Deswegen wird die Information nicht in einer nach MVD-BEA exportierten IFC-Datei mit enthalten.

Anhang D

Compact disc

Auf der beigefuegten CD befindet sich folgender Inhalt:

- Diese Dokumantation als PDF
- Das IBPSBuilding-Package in Modelica-Sprache
- Die Dokumentation MVD-BEA
- Quellcodes der EIX-Bibliothek in Java
- IFC-Dateien für Tests des Datenabrufs mittels EIX

Glossar

.NET	eine Softwareentwicklungsplattform, die aus einer Laufzeitumgebung und einer Sammlung von Klassenbibliothek und Schnittstellen besteht.
2D	Zweidimensional
3D	Dreidimensional
3ds	eines von Autodesk 3ds Max genutztes Dateiformat für 3D-Modellierung
API	Application Programming Interface, dt. Schnittstelle zur Anwendungsprogrammierung
BEA	Design to Building Energy Analysis, ein offen zugängliches MVD
BIM	Building Information Modeling
bSI	buildingSMART International, gegründet als Industrieallianz für Interoperabilität e.V.(IAI) 1995, mit dem Ziel, die Projektabwicklung mittels effizienter Methoden integrierter Informationsverarbeitung durchgängiger und effektiver zu gestalten
C	eine in den frühen 1970er entwickelte imperative Programmiersprache
C++	eine von der ISO genormte Programmiersprache, die als Erweiterung der C-Sprache entwickelt und Objektorientierte Programmierung unterstützt
COBie	Construction Operations Building Information Exchange, ein Dateiformat für die Dokumentation non-geometrischer Daten aus BIM
CPU	Central Processing Unit
C#	eine von Microsoft im Rahmen der .NET-Plattform entwickelte Programmiersprache
CSV	Character-Separated Values, ein Dateiformat zur Beschreibung strukturierter Daten, die mit Zeichen getrennt werden
DGNB	Deutsches Gütesiegel Nachhaltiges Bauen
dxf	Drawing Exchange Format, ein CAD-Dateiformat von Autodesk
Eclipse	eine quelloffene integrierte Entwicklungsumgebung hauptsächlich für Java
EIX	Energy Information EXchange
EMP	IFC Extended Material Properties
ERM	Exchange Requirements Models
EXPRESS	eine Modellierungssprache für STEP-Produktdatenmodelle

FMI	Functional Mock-up Interface, ein Standard für Datenaustausch und Simulationskopplung auf mehreren Plattformen
FORTRAN	FORmula TRANslation, eine Programmiersprache
gbXML	Green Building Extended Markup Language
GUI	Graphical User Interface, dt. 'Grafische Benutzeroberfläche'
GUID	Globally Unique ID
IBPS	Implementing a Building Performance Simulation, ein Lehrseminar an der TU München
ID	Identifikator, Kennung
IDF	Input Data Format, ein Dateiformat für Eingabeinformationen in EnergyPlus
IDM	Information Delivery Manual
IFC	Industry Foundation Classes
Java	eine objektorientierte Programmiersprache unter der Marke von Sun Microsystems (2010 von Oracle gekauft)
JDK	Java-Development-Kit
LBNL	Lawrence Berkeley National Laboratory, eine Forschungseinrichtung vom United States Department of Energy
LOD	Level of Detail, dt. Detaillierungsgrad, beschreibt verschiedenen Detailstufen bei der Darstellung virtueller Welten
MVD	Model View Definition
P-set	Property-set
STEP	Standard for the Exchange of Product model data, ein Standard nach ISO 10303 zur Beschreibung von Produktdaten
TGA	Technische Gebäudeausrüstung
TMY	Typical Meteorological Year, charakteristische Jahreswetterdaten von einer Klimadatenbank, die von mehreren Jahren generiert werden kann
VBA	Visual Basic for Applications, eine Skriptsprache für die Steuerung von Abläufen der Microsoft-Office-Programmfamilie
WDV	Wärmedämmverbundsystem
WebGL	Web Graphics Library, dt. 'Web-Grafik-Bibliothek', ein Bestandteil von Webbrowern, mit dem 3D-Grafiken direkt im Browser dargestellt werden können
xBIM	eXtensible Building Information Modelling
XML	Extensible Markup Language, dt. 'erweiterbare Auszeichnungssprache'

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literaturverzeichnis

Association, M. (9 Mai, 2012). Modelica - A Unified Object-Oriented Language for Systems Modeling Version 3.3. URL: https://modelica.org/documents/ModelicaSpec33.pdf

BMVBS (2013). *Leitfaden nachhaltiges Bauen.* Berlin: Bundesministerium für Verkehr Bau und Stadtentwicklung.

buildingSMART International (03.02.2014). IFC Overview summary. URL: http://www.buildingsmart-tech.org/specifications/ifc-overview/ifc-overview-summary

buildingSMART International (28.02.2013). IFC4 Documentation. URL: http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm

Clarke, J. A. (2001). *Energy simulation in building design* (2nd ed Aufl.). Oxford: Butterworth-Heinemann.

DGNB (2008). Das Deutsche Gütesiegel Nachhaltiges Bauen: Aufbau-Anwendung-Kriterien.

Eastman, C., Teicholz, P., Sacks, R. & Liston, K. (2011). *BIM Handbook: A guide to building information modeling for owners, managers, designers, engineers, and contractors* (2 Aufl.). Hoboken and N.J: John Wiley & Sons.

Google Project Hosting (14.02.2014). ifcplusplus: Open Source C++ implementation of IFC. URL: https://code.google.com/p/ifcplusplus/

Häfele, K.-H. (April, 2009). IFC Implementation Agreement Space Boundary: Overview on the common agreements for implementing space boundaries.

ITI GmbH (2013). SimulationX.

Maile, T., O'Donnell, J., Bazjanac, V. & Rose, C. (2013). BIM-Geometry Modelling Guidelines for Building Energy Performance Simulation. In: *13th Conference of International Building Performance Simulation Association.*

Modelica Association (19.01.2014). Homepage Modelica and the Modelica Association. URL: https://modelica.org/

Nasyrov, V. (22.Nov 2013). *Building Information Models als Input für energetische Gebäudesimulation*. Dissertation, Technische Universität München, München.

National Institute of Building Sciences (2012). National BIM Standard - United States Version 2. URL: http://www.nationalbimstandard.org/nbims-us-v2/pdf/NBIMS-US2_c4.4.pdf

Olsson, H. (2005). External Interface to Modelica in Dymola. In: G. Schmitz (Hrsg.), *Proceedings of the 4th International Modelica Conference*, S. 603–611. the Modelica Association.

OpenBIM (14.02.2014). The xBIM Toolkit: the open toolbox for BIM. URL: http://www.openbim.org/

Page, J. (01.02.2014). Math Open Reference: Area of any polygon (Coordinate Geometry). URL: http://www.mathopenref.com/coordpolygonarea.html

See, R. (2011). Design to Building Energy Analysis: IFC Release Specific AEC/FM View Description (IFC 2x3). URL: http://www.blis-project.org/IAI-MVD/MVDs/GSA-003/

See, R., Karlshøj, J. & Davis, D. (2012). An Intergrated Process for Delivering IFC Based Data Exchange.

Tauscher, E. (2013). IFC TOOLS Project. URL: http://www.ifctoolsproject.com/

Theiler, M. (30.Mai, 2013). Getting Started - Java Toolbox IFC2x3/IFC4. URL: http://www.ifctoolsproject.com/download.html

Tiller, M. (2005). Implementation of a Generic Data Retrieval API for Modelica. In: G. Schmitz (Hrsg.), *Proceedings of the 4th International Modelica Conference*. the Modelica Association.

University Oslo (2003). Lecture 3: Coordinate Systems and Transformations.

van Treeck, C. (2011). Lecture Notes Building Energy Modeling and Simulation. München.

Wetter, M. (05.03.2014). Modelica Buildings Library. URL: http://simulationresearch.lbl.gov/modelica/

Yan, W., Clayton, M., Haberl, J., Jeong, W., Kim, J. B., Kota, S., Alcocer, J. L. B. & Dixit, M. (August 2013). Interfaceing BIM with Building Thermal and Daylighting Modeling. Chambéry and France.