Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

DIPLOMARBEIT

Master Thesis

Entwicklung einer hybriden Simulationsumgebung in Modelica zur Kopplung von dynamischer Gebäudesimulation an vereinfachte Verfahren zur Ermittlung von Energiekennzahlen

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von

Ao.Univ.Prof. Dipl. Ing. Dr. Thomas Bednar
E206 Institut für Hochbau und Technologie

eingereicht an der Technischen Universität Wien
Fakultät für Bauingenieurwesen

von

Hannes Edinger
Mat.Nr. 9926077
Meldemannstr. 22/2, 1200 Wien

Wien, am 05. Dezember 2010

Abstract

Subject-matter of this master thesis is the development of a hybrid simulation framework which allows coupling of simplified methods for calculation of building performance coefficients and methods of dynamic building-simulation, based on one unified building-model and calculation-process.

This paper aims to illustrate the developed software for the user as well as for future developers. Furthermore practical Modelica-knowledge, which was gained during the development of the software, and which may be interesting for future developers, is presented.

Using the developed framework, the Austrian Standards for calculation of the heating- and cooling-demand of buildings where implemented and tested. An exemplary calculation demonstrates the coupling of these simplified methods of energy-balancing with dynamically modeled building elements.

Zusammenfassung

Im Rahmen der Arbeit wurde eine hybride Simulationsumgebung in Modelica entwickelt, die es auf Basis eines vereinheitlichten Gebäudemodells bzw. Berechnungsablaufes ermöglicht, vereinfachte Verfahren zur Ermittlung von Energiekennzahlen und Elemente aus der dynamischen Gebäudesimulation zu koppeln.

Mit dem vorliegenden Text ist die entwickelte Software bis ins Detail sowohl für Anwender wie auch für eventuelle spätere Weiterentwicklung dokumentiert. Für eine zukünftige Weiterentwicklung sind außerdem die praktischen Erfahrungen, welche der Autor im Umgang mit verschiedenen Modelica-Umgebungen gemacht hat, zusammengefasst.

Unter Verwendung des entwickelten Frameworks wurden die ÖNORMEN der Serie B 8110 zur Ermittlung von Heizwärme- und Kühlbedarf implementiert und getestet. Weiters wurde eine exemplarische Berechnung, bei der die Rechenverfahren dieser Normen mit Elementen der dynamischen Gebäudesimulation in einem hybriden Modell gekoppelt wurden, durchgeführt.

Danksagung

Danke an **meine Eltern**, dafür, dass sie mir am Anfang meines Studiums eine eigene Wohnung in Wien sowie ein regelmäßiges Einkommen zur Verfügung gestellt haben, und – mit einem Augenzwinkern – dafür, dass sie später irgendwann aufgehört haben, zu fragen, wann ich endlich mit dem Studium fertig sein werde.

Danke an das Unternehmen **Vasko + Partner Ingenieure**, dafür, dass es mir Gelegenheit gegeben hat, diese Arbeit im Rahmen eines von ihm initiierten Forschungsprojektes zu schreiben, und dafür, dass man mich für die Zeit, welche für den Abschluss meines Studiums nötig war, von meiner regulären Tätigkeit als Bauphysiker freigestellt hat.

Nicht zuletzt auch Danke an **Ao.Univ.Prof. Dipl. Ing. Dr. Thomas Bednar**, dafür, dass er mich und das Projekt mit seiner umfangreichen fachlichen Kompetenz unterstützt hat, und dafür, dass er als Diplomarbeits-Betreuer und Bauphysik-Professor stets einen sehr kollegialen und freundlichen Arbeitsstil gepflegt hat.

Inhaltsverzeichnis

1 Einleitung 1

1.1 Gegenstand und Kontext der Arbeit 1

1.1.1 Gegenstand 1

1.1.2 Abgrenzung 1

1.1.3 Kontext 1

1.1.4 Projektkurzdarstellung 2

1.2 Text und Quelltext 2

1.2.1 Textauszeichnungen und Symbole 2

1.2.2 Quelltext 3

1.2.3 Copyrights, Markenzeichen, Credits, etc. 4

2 Grundlagen 5

2.1 Energiekennzahlen und deren Berechnung 5

2.1.1 Allgemeines 5

2.1.2 Kategorisierung von Berechnungs- bzw. Simulationsmodellen . 6

2.1.3 Aktuelle Berechnungsverfahren 7

2.1.4 Zukunftsaussichten 13

2.1.5 Warum Kopplung von vereinfachten und simulativen Methoden? 15

2.2 Modelica 16

2.2.1 Vor- und Nachteile von Modelica 16

2.2.2 OpenModelica 21

2.2.3 Kommerzielle Modelica-Umgebungen 23

2.2.4 Zusammenfassung 27

3 Umsetzung 29

3.1 Grundsätzliches 29

3.1.1 Kombination unterschiedlicher Zeitschrittweiten: Hybride Zeitgitter 29

3.1.2 Kombination diskret-kontinuierlich, stationär-instationär bzw. kausal-akausal: Hybride Modelle . . . 33
3.1.3 Typischer Aufbau von Norm-Rechenverfahren . . . 34
3.2 Grundelemente der Simulationsumgebung . . . 35
3.2.1 Rechenschritte . . . 37
3.2.2 Formeln (Gleichungen) . . . 38
3.2.3 Tabellen . . . 42
3.2.4 Sonstiges . . . 52

4 Anwendung 55
4.1 Die ÖNORM B 8110-5 . . . 55
4.2 Die ÖNORM B 8110-6 . . . 55
4.2.1 Wärmeverluste Q_l . . . 56
4.2.2 Wärmegewinne Q_g . . . 62
4.3 Numerisches Anwendungsbeispiel . . . 64
4.3.1 Details des exemplarischen Gebäudemodells . . . 65
4.3.2 Details zur Konvertierung zwischen Stunden- und Monatszeitgitter . . . 68
4.3.3 Zusammenführung in ein hybrides Modell . . . 71

A Hinweise zur Entwicklung mit Modelica 75
A.1 Allgemeines . . . 75
A.2 Hinweise zu den untersuchten Modelica-Umgebungen . . . 76
A.2.1 OpenModelica . . . 76
A.2.2 MathModelica & Dymola . . . 77
A.3 Konkrete Bugs & fehlende v3.0-Features in den untersuchten Modelica-Umgebungen . . . 77
A.4 Was man anders hätte machen können . . . 85

Literaturverzeichnis 86

Abbildungsverzeichnis 88

Tabellenverzeichnis 90

1 Einleitung

1.1 Gegenstand und Kontext der Arbeit

1.1.1 Gegenstand

Gegenstand der Diplomarbeit ist die Entwicklung einer hybriden Simulationsumgebung in Modelica zur Kopplung von dynamischer Gebäudesimulation an vereinfachte Verfahren zur Ermittlung von Energiekennzahlen, sowie die Dokumentation des entwickelten Systems. Implementiert werden im Rahmen der Diplomarbeit die Verfahren der Normenserie ÖNORM B 8110. Darüber hinaus wurde ein dynamisches Modell eines unkonditionierten Pufferraumes erstellt, an das mehrere Zonen gekoppelt werden können. Weiters wurde eine Schnittstelle geschaffen, die Klimadaten aus Testreferenzjahren im entwickelten Framework verfügbar macht.

1.1.2 Abgrenzung

Nicht Gegenstand der Arbeit ist die Implementation der Normenserie ÖNORM H 5055 bis H 5059 – diese wird parallel zur Implementation der Normenserie ÖNORM B 8110 von anderen Projektbeteiligten vorgenommen. Die Anwendung des entwickelten Systems auf reale Objekte ist ebenfalls nicht Gegenstand dieser Arbeit.

1.1.3 Kontext

Diese Arbeit wurde im Rahmen des Forschungsprojektes *Bestandsimmobilien: Energieausweis – realer Energieverbrauch – Optimierung* der

Vasko + Partner Ingenieure
Ziviltechniker für Bauwesen und Verfahrenstechnik GmbH
Grinzinger Allee 3

1190 Wien
Firmenbuchnummer: FN 112104 i HG Wien

mit Finanzierung aus Fördermitteln der *ZIT – Zentrum für Innovation und Technologie GmbH* im Rahmen des Call *Vienna Environment 2008* erstellt (ID des Antrags: 352330). Kooperationspartner des Projektes ist die *Post & Telekom Immobiliengesellschaft m.b.H.* (PTI). Auftragnehmer sind die *TU Wien, Zentrum für Bauphysik und Bauakustik* sowie die *Wiener Magistratsabteilung 39*.

1.1.4 Projektkurzdarstellung

Die Entwicklung des Berechnungstools zielt darauf ab, Bestandsimmobilien zu erfassen und anhand anerkannter Rechenregeln (Energieausweis) als Rechenmodell darzustellen. Mit den bekannten tatsächlichen Verbräuchen kann das Rechenmodell geprüft bzw. „kalibriert“ werden (Vergleich Soll-/Ist-Verbrauch).

In weiterer Folge können Maßnahmen zur Verbesserung EDV-unterstützt berechnet werden. Durch Zugrundlegung von Kosten für einzelne Maßnahmen kann so eine umfassende Bewertung (Ist-Situation; mögliche Verbesserungsmaßnahmen inkl. Amortisationszeiten) der Immobilien – auf Basis eines anerkannten Modells (Energieausweis) – erreicht werden. Durch den Soll/Ist-Vergleich der Verbräuche werden darüber hinaus die derzeit anerkannten Rechenregeln auf ihre Praxistauglichkeit geprüft. Anhand einer bestimmten Anzahl von Objekten aus dem Immobilien Portfolio der PTI wird das Softwaremodul getestet.

1.2 Text und Quelltext

1.2.1 Textauszeichnungen und Symbole

Der Text dieser Diplomarbeit sollte auch als Dokumentation bzw. „Manual“ zur erstellten Software dienen. Er wurde daher mit Textauszeichnungen und Symbolen versehen, welche die Lesbarkeit des Textes erleichtern sollen.

- Besonders *betonte Textteile* werden *kursiv* gedruckt.
- *Eigennamen* oder *sonstige Bezeichner* werden *schräggestellt* gedruckt.

- Wenn im Fließtext auf *Identifier* im Sourcecode verwiesen wird, so werden diese *in serifenloser, kursiver, dunkelroter Schrift* dargestellt. Im Normalfall werden keine Fully Qualified Identifier im Text verwendet.
- Wenn im Fließtext `Modelica Code` eingebettet ist, so wird dieser `mit fester Zeichenbreite` (Teletype-Schriftart) gedruckt.
- Hinweise, die der Autor für die Arbeit mit der erstellten Software für besonders wichtig erachtet, werden mit „☞ **Hinweis:** “ eingeleitet.
- Querverweise innerhalb des Textes werden mit einem ➪ Pfeil-Symbol gekennzeichnet.
- Verweise aufs Literaturverzeichnis bestehen aus den ersten drei Buchstaben des Autorennamens und den letzten beiden Ziffern des Erscheinungsjahres in eckigen Klammern (z.B.: [Hom06]).
- Querverweise innerhalb des Dokuments und URLs bzw. Verweise auf Internet-Seiten werden dunkelblau gedruckt und können in der digitalen Version des Dokuments als Link angeklickt werden.

Abbildungen, Tabellen und Sourcecode-Listings werden wie üblich unterhalb beschriftet und nummeriert. Listings werden darüber hinaus grau hinterlegt und links neben dem Sourcecode mit Zeilennummern versehen:

```
1 model HelloModelica "A differential equation"
2   Real x(start=1);
3   equation
4   der(x)=-x;
5 end HelloModelica;
```

Listing 1.1: Hello DAE

1.2.2 Quelltext

Teil dieser Diplomarbeit ist ein Datenträger, der Folgendes enthält:

- Modelica Sourcecode des entwickelten Frameworks
- Modelica Sourcecode des Modells der ÖNorm B8110-5
- Modelica Sourcecode des Modells der ÖNorm B8110-6

1.2.3 Copyrights, Markenzeichen, Credits, etc.

- *Modelica®* ist ein eingetragenes Markenzeichen der *Modelica Association*.
- *MathModelica®* ist ein eingetragenes Markenzeichen der *MathCore Engineering AB*
- *Dymola®* ist ein eingetragenes Markenzeichen der *Dassault Systemes AB*
- *Matlab®* und *Simulink®* sind eingetragene Markenzeichen der *The MathWorks, Inc.*

Dieses Dokument und alle enthaltenen Vektorgrafiken wurden mit LaTeX 2_ϵ erstellt. Eine digitale Version der Arbeit ist an der Universitätsbibliothek der TU Wien erhältlich. Die Verwendung von Abbildungen aus diesem Dokument unterliegt den Bestimmungen der © Creative Commons Share Alike Lizenz 3.0 [Cre10].

Urheber- und Nutzungsrechte an der erstellten Software haben ausschließlich die Vasko + Partner GmbH sowie der Autor. Jede Verwendung der Software oder von Teilen der Software (im Quelltext oder in kompilierter Form) durch Dritte bedarf der ausdrücklichen schriftlichen Zustimmung dieser zwei Parteien.

2 Grundlagen

2.1 Energiekennzahlen und deren Berechnung

2.1.1 Allgemeines

Im allgemeinsten Sinn kann eine Energiekennzahl als einheitsbezogener *Vergleichswert* beschrieben werden, welcher den Energiebedarf eines Objektes *unter standardisierten Bedingungen* angibt.

Energiekennzahlen für Gebäude charakterisieren ein Gebäude incl. seiner gebäudetechnischen Anlagen hinsichtlich seiner Energieeffizienz. Sie sind vielfach Grundlage für die Vergabe von öffentlichen Fördermitteln, und seit 2008 schreibt der Gesetzgeber in Österreich die Vorlage eines „Ausweises der Gesamtenergieeffizienz eines Gebäudes“ – dem sog. Energieausweis – beim Verkauf und bei der In-Bestand-Gabe von Gebäuden und Nutzungsobjekten vor [Bun06].

Gebäudeenergiekennzahlen haben nur begrenzte Aussagekraft über den tatsächlichen Energieverbrauch und beinhalten auch *keine expliziten Sicherheitsfaktoren*. Deshalb sind sie z.B. nicht zur Dimensionierung von gebäudetechnischen Anlagen oder für Wirtschaftlichkeits-Analysen geeignet. Relevante Diskrepanzen zwischen dem physikalischen Modell zur Ermittlung von Gebäudeenergiekennzahlen und der Realität entstehen u.a. wegen

- Abweichungen im tatsächlichen Nutzerverhalten gegenüber dem standardisierten Nutzerverhalten,
- Abweichung in der tatsächlichen gebäudetechnischen Steuerung bzw. Regelung gegenüber deren stark abstrahierter Abbildung in den Gebäudetechnik-Normen (➪ siehe Abschnitt 2.1.3.1b),
- Abweichungen im lokalen Mikroklima gegenüber dem standardisierten Klimamodell

- Abweichungen der tatsächlichen baulichen Situation vom abstrahierten Gebäudemodell.

2.1.2 Kategorisierung von Berechnungs- bzw. Simulationsmodellen

Berechnungs- bzw. Simulationsmodelle können nach ihren Eigenschaften klassifiziert werden in statische bzw. dynamische, diskrete bzw. stetige, deterministische bzw. stochastische Modelle:

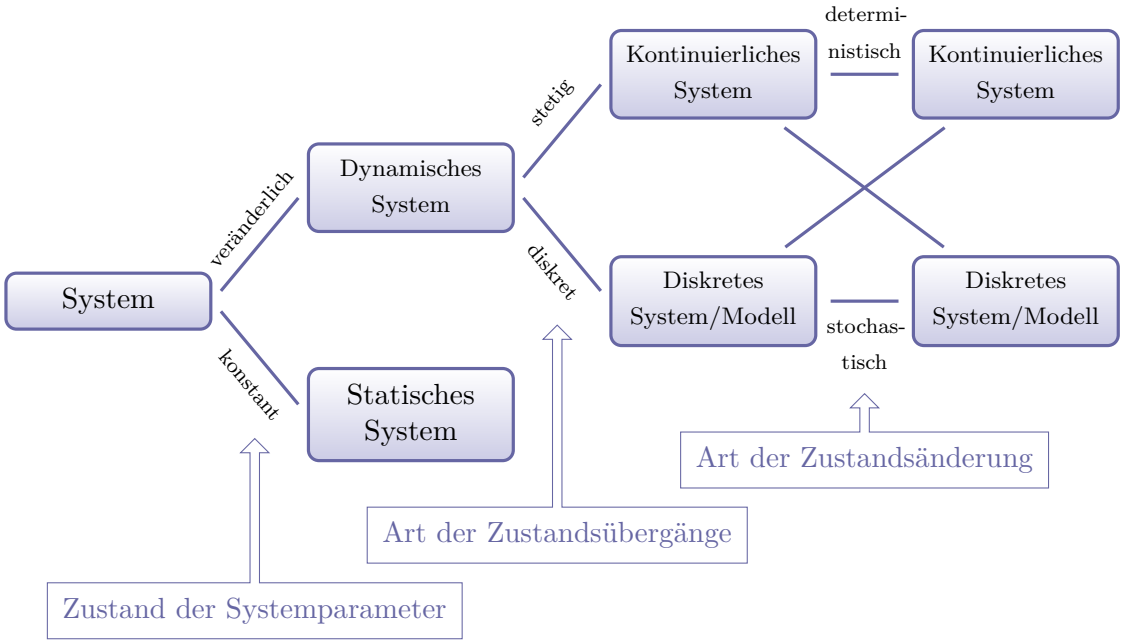


Abbildung 2.1: Kategorisierung von Berechnungs- bzw. Simulationsmodellen

Bei der Berechnung von Energiekennzahlen nach den in ➪ Abschnitt 2.1.3.1 angeführten ÖNORMEN kommt ein diskretes Verfahren ohne stochastische Elemente zum Einsatz. Dynamische Gebäudesimulation basiert dagegen üblicherweise auf kontinuierlichen Modellen mit diskreten (teilweise auch stochastischen) Eingangswerten. Die Zustandsübergänge können diskret (z.B. Aktivierung von Sonnenschutz) oder kontinuierlich (z.B. aus TRY-Daten interpolierter Außentemperaturverlauf) sein. Die Art der Zustandsübergänge kann deterministisch oder stochastisch sein.

2.1.3 Aktuelle Berechnungsverfahren

2.1.3.1 Bilanzverfahren mit Monatsintervallen

Die Berechnung von Energiekennzahlen ist in Österreich derzeit (primär) in den nachfolgend aufgelisteten Normen geregelt.

Die diesen Normen zugrunde liegenden physikalischen Modelle sind quasi-*stationäre* Monatsbilanzverfahren, d.h. es wird eine Serie von 12 Bilanzen so berechnet, als würde das System jeweils einen Monat lang im thermodynamischen Gleichgewicht stehen, und an jedem Monatsübergang eine sprunghafte Änderungen in einen anderen Gleichgewichtszustand erfahren. Die Bilanzierungsverfahren sind monatsweise *diskret*, d.h. die Ergebnisse der einzelnen Monate sind voneinander völlig unabhängig.

a) Bautechnik-Normenserie ÖNORM B 8110-6:

Norm	Titel
ÖNORM B 8110-1	Wärmeschutz im Hochbau – Teil 1: Anforderungen an den Wärmeschutz und Deklaration des Wärmeschutzes von Gebäuden/Gebäudeteilen - Heizwärmebedarf und Kühlbedarf
ÖNORM B 8110-5	Wärmeschutz im Hochbau – Teil 5: Klimamodell und Nutzungsprofile
ÖNORM B 8110-6	Wärmeschutz im Hochbau – Teil 6: Grundlagen und Nachweisverfahren - Heizwärmebedarf und Kühlbedarf

Tabelle 2.1: Bautechnische Normen in der Energiekennzahl-Ermittlung

Die ÖNORM B 8110-6 gliedert sich in vereinfachte und detaillierte Berechnungsverfahren, wobei sich die detaillierten Verfahren im Wesentlichen an jenen der EN ISO 13790 – Energieeffizienz von Gebäuden - Berechnung des Energiebedarfs für Heizung und Kühlung (ISO 13790:2008) – orientieren. Wie oben erwähnt beschreibt diese Norm ein Berechnungsverfahren in 12 diskreten Zeitschritten. Eine der wesentlichen Verbesserungen dieses Monatsbilanzverfahrens (gegenüber dem Heizgradtag-Verfahren) ist die Einführung von Ausnutzungsgraden für Gewinne.
Die vereinfachten Verfahren zur Leitwert-Ermittlung mit Temperaturkorrekturfaktoren nach ÖNORM B 8110-6 liefern in der Regel ungenauere Ergebnisse als jene der detaillierten Ermittlung und werden in der gegenständlichen Arbeit (zum Teil) nicht näher betrachtet.

Während die Verfahren nach ÖNORM B 8110-6 durchaus zuverlässige Werte für den HWB liefern, kann der Fehler in der KB-Berechnung bis zu 40% betragen [Sof09].

b) Haustechnik-Normenserie ÖNORM H 5055 bis H 5059:

Norm	Titel
ÖNORM H 5055	Energieausweis für Gebäude
ÖNORM H 5056	Gesamtenergieeffizienz von Gebäuden – Heiztechnik-Energiebedarf
ÖNORM H 5057	Gesamtenergieeffizienz von Gebäuden – Raumlufttechnik-Energiebedarf für Wohn- und Nichtwohngebäude
ÖNORM H 5058	Gesamtenergieeffizienz von Gebäuden – Kühltechnik-Energiebedarf
ÖNORM H 5059	Gesamtenergieeffizienz von Gebäuden - Beleuchtungsenergiebedarf (Nationale Ergänzung zu ÖNORM EN 15193)

Tabelle 2.2: Gebäudetechnische Normen in der Energiekennzahl-Ermittlung

Ein grundlegendes Problem in der aktuellen Fassung der Normenserie ÖNORM H 5055 bis H 5059 besteht darin, dass sich Gebäudenutzung und gebäudetechnische Steuerungen aufgrund des groben Zeitrasters der Berechnungsmethode nur stark abstrahiert abbilden lassen. In der Gebäudetechnik-Normenserie werden zahlreiche „Beiwerte“, „Mittelwerte“ und „Hilfswerte“ eingeführt, mit deren Hilfe versucht wird, diese Phänomene abzubilden. Ähnlich wie bei den Beiwerten der vereinfachten Verfahren der ÖNORM B8110-6 (Temperaturkorrekturfaktoren) hat sich gezeigt, dass mit dieser Methode nicht immer zufriedenstellende Ergebnisse erzielt werden können. Um das Nutzerverhalten und die gebäudetechnischen Steuerungen/Regelungen besser abbilden zu können, ist ein Verfahren mit feinerem Zeitraster nötig.

Eine umfassende Darstellung der in a) und b) angeführten Normen sowie der rechtlichen Situation „rund um Energieausweise“ in Österreich findet sich auch in [Pöh07].

c) Sonstige Normen:

In den in a) und b) angeführten Normen wird auf folgende weitere Normen Bezug genommen:

Norm	Titel
ÖNORM B 1800	Ermittlung von Flächen und Rauminhalten von Bauwerken
ÖNORM B 8110-2	Wärmeschutz im Hochbau – Teil 2: Wasserdampfdiffusion und Kondensationsschutz
ÖNORM B 8110-3	Wärmeschutz im Hochbau – Wärmespeicherung und Sonneneinflüsse
ÖNORM EN 12831	Heizungsanlagen in Gebäuden – Verfahren zur Berechnung der Norm-Heizlast

weiter auf der nächsten Seite

ÖNORM EN 12975-1	Thermische Solaranlagen und ihre Bauteile – Kollektoren – Teil 1: Allgemeine Anforderungen
ÖNORM EN 13141-7	Lüftung von Gebäuden – Leistungsprüfungen von Bauteilen/Produkten für die Lüftung von Wohnungen – Teil 7: Leistungsprüfung von mechanischen Zuluft- und Ablufteinheiten (einschließlich Wärmerückgewinnung) für mechanische Lüftungsanlagen in Einfamilienhäusern
ÖNORM EN 13779	Lüftung von Nichtwohngebäuden – Allgemeine Grundlagen und Anforderungen für Lüftungs- und Klimaanlagen und Raumkühlsysteme
ÖNORM EN 13829	Wärmetechnisches Verhalten von Gebäuden – Bestimmung der Luftdurchlässigkeit von Gebäuden – Differenzdruckverfahren (ISO 9972:1996, modifiziert)
ÖNORM EN 13947	Wärmetechnisches Verhalten von Vorhangfassaden – Berechnung des Wärmedurchgangskoeffizienten
ÖNORM EN 14511 (alle Teile)	Luftkonditionierer, Flüssigkeitskühlsätze und Wärmepumpen mit elektrisch angetriebenen Verdichtern für die Raumheizung und -kühlung
ÖNORM EN 15193	Energetische Bewertung von Gebäuden – Energetische Anforderungen an die Beleuchtung
ÖNORM EN 15316-4-3	Heizungsanlagen in Gebäuden – Verfahren zur Berechnung der Energieanforderungen und Nutzungsgrade der Anlagen – Teil 4-3: Wärmeerzeugungssysteme & thermische Solaranlagen
ÖNORM EN 15879-1	Prüfung und Leistungsbemessung von Direktaustausch-Erdreichwärmepumpen mit elektrisch angetriebenen Verdichtern zur Raumbeheizung und/oder -kühlung – Teil 1: Direktaustausch/Wasser-Wärmepumpe
ÖNORM EN 255-3	Luftkonditionierer, Flüssigkeitskühlsätze und Wärmepumpen mit elektrisch angetriebenen Verdichtern – Heizen – Prüfungen und Anforderungen an die Kennzeichnung von Geräten zum Erwärmen von Brauchwarmwasser
ÖNORM EN 308	Wärmeaustauscher – Prüfverfahren zur Bestimmung der Leistungskriterien von Luft/Luft und Luft/Abgas-Wärmerückgewinnungsanlagen
ÖNORM EN 410	Glas im Bauwesen – Bestimmung der lichttechnischen und strahlungsphysikalischen Kenngrößen von Verglasungen
ÖNORM EN ISO 10077-1	Wärmetechnisches Verhalten von Fenstern & Türen und Abschlüssen – Berechnung des Wärmedurchgangskoeffizienten – Teil 1: Allgemeines (ISO 10077-1:2006)

weiter auf der nächsten Seite

ÖNORM EN ISO 10211	Wärmebrücken im Hochbau – Wärmeströme und Oberflächentemperaturen – Detaillierte Berechnungen (ISO 10211:2007)
ÖNORM EN ISO 13370	Wärmetechnisches Verhalten von Gebäuden – Wärmeübertragung über das Erdreich – Berechnungsverfahren (ISO 13370:2007)
ÖNORM EN ISO 13789	Wärmetechnisches Verhalten von Gebäuden – Spezifischer Transmissions- und Lüftungswärmedurchgangskoeffizient – Berechnungsverfahren (ISO 13789:2007)
ÖNORM EN ISO 13790	Energieeffizienz von Gebäuden – Berechnung des Energiebedarfs für Heizung und Kühlung (ISO 13790:2008)
ÖNORM EN ISO 14683	Wärmebrücken im Hochbau – Längenbezogener Wärmedurchgangskoeffizient – Vereinfachte Verfahren und Anhaltswerte (ISO 14683:2007)
ÖNORM EN ISO 6946	Bauteile – Wärmedurchlasswiderstand und Wärmedurchgangskoeffizient – Berechnungsverfahren (ISO 6946:2007)
ÖNORM H 6038	Lüftungstechnische Anlagen – Kontrollierte mechanische Be- und Entlüftung von Wohnungen mit Wärmerückgewinnung – Planung, Montage, Prüfung, Betrieb und Wartung
ÖNORM H 7500	VORNORM Heizungssysteme in Gebäuden – Verfahren zur Berechnung der Norm-Heizlast (Nationale Ergänzung zu ÖNORM EN 12831)
ÖNORM M 7701	Sonnentechnische Anlagen – Näherungsverfahren zur Bemessung von Flachkollektoren in Warmwasser-Aufbereitungsanlagen
[keine Kurzbezeichnung]	Österreichisches Institut für Bautechnik (OIB) (Hrsg.), Leitfaden Energietechnisches Verhalten von Gebäu-den. Version 2.6, April 2007
2002/31/EG	Richtlinie 2002/31/EG der Kommission vom 22. März 2002 zur Durchführung der Richtlinie 92/75/EWG des Rates betreffend die Energieetikettierung für Raumklimageräte
DIN V 18599-2	VORNORM Energetische Bewertung von Gebäuden – Berechnung des Nutz-, End- und Primärenergiebedarfs für Heizung, Kühlung, Lüftung, Trinkwarmwasser und Beleuchtung – Teil 2: Nutzenergiebedarf für Heizen und Kühlen von Gebäudezonen
ON-V 31	Bauwesen 1 – Katalog für wärmeschutztechnische Rechenwerte von Baustoffen und Bauteilen

Tabelle 2.3: Normen mit Bezug zu den Serien ÖNORM B 8110-1/5/6 und ÖNORM H 5055-5059

Diese Normen stellen Verfahren zur Verfügung, die zur Ermittlung verschiedener

Eingangsgrößen für die Berechnung von Energiekennzahlen nach ÖNORM B 8110 bzw. ÖNORM H5055-59 nötig sind, und verweisen zum Teil selbst wieder auf weitere Normen, die hier nicht mehr angeführt sind.

2.1.3.2 Bilanzverfahren mit Stundenintervallen

Eine der interessantesten Änderungen in der ÖNORM B 8110-5 gegenüber ihrer Vorgängerversion von 2007 ist die Einführung des halbsynthetischen Klimadatenmodells, auf dessen Basis stündliche Temperatur- und Luftfeuchtewerte generiert werden können. Die Temperaturen sind insofern mit den Monatsmittelwerten aus dem Dreischichten-Regressionsmodell kompatibel, als der Algorithmus des halbsynthetischen Klimadatenmodells mittelwertstreu ist. Dadurch kann der Heizwärmebedarf weiterhin im Monatsbilanzverfahren ermittelt werden, während für jene Bereiche, die im groben Zeitraster des Monatsbilanzverfahrens nicht ausreichend genau abgebildet werden können, verbesserte Bilanzierungsverfahren auf Basis des Stunden-Zeitrasters entwickelt werden können. Der Trend ist also offensichtlich.

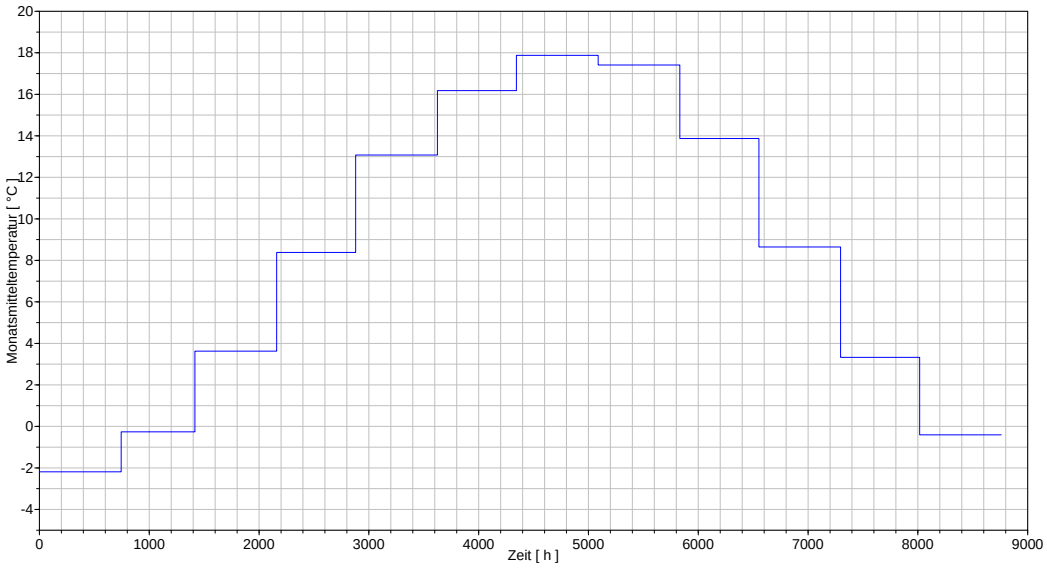


Abbildung 2.2: Jahresverlauf der Monatsmitteltemperaturen für Wien (Innere Stadt) nach ÖNORM B 8110-5 (Dreischichten-Regressionsmodell)

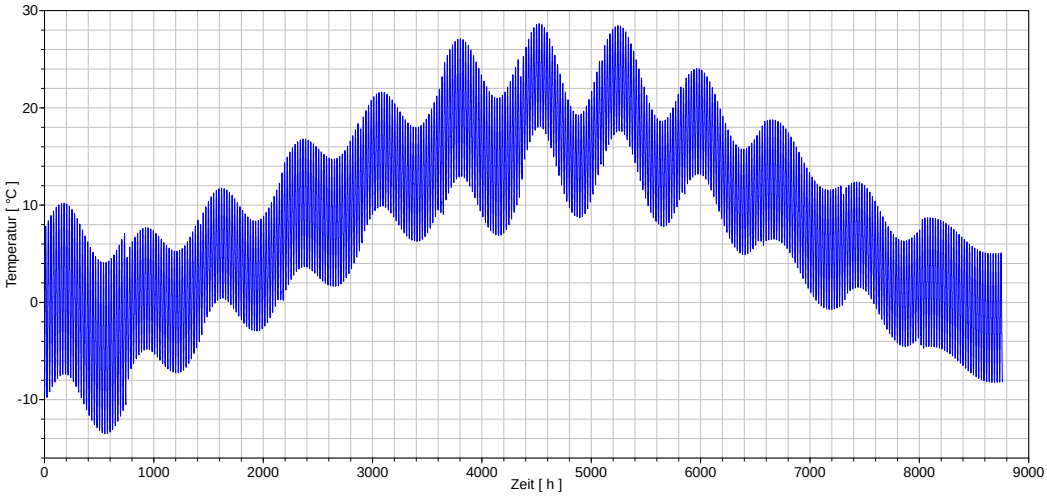


Abbildung 2.3: Jahresverlauf der Stunden-Mitteltemperaturen für Wien (Innere Stadt) nach ÖNORM B 8110-5 (halbsynthetisches Klimadatenmodell)

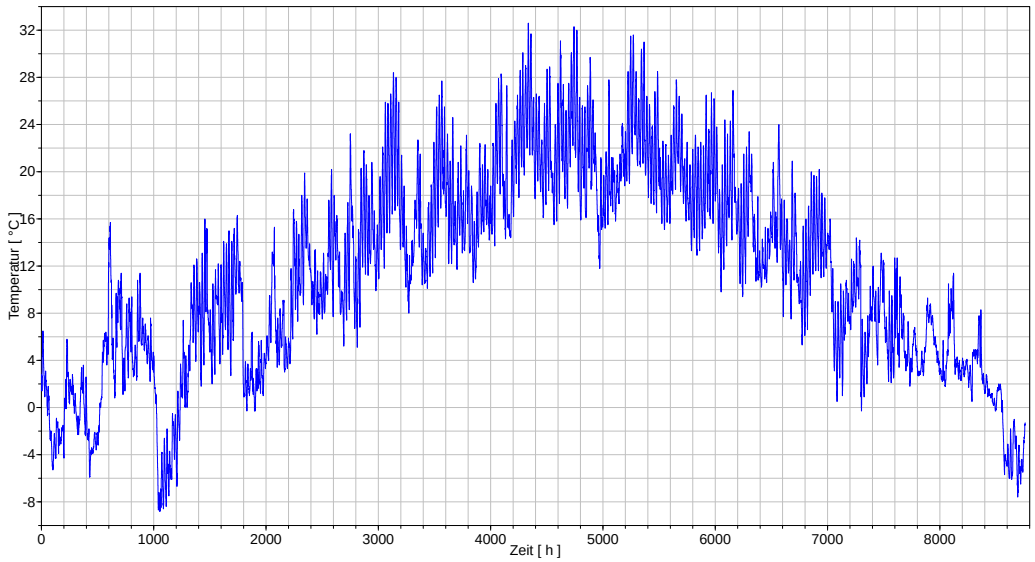


Abbildung 2.4: Jahresverlauf der Stunden-Mitteltemperaturen des Testreferenzjahres für Wien (Innere Stadt)

2.1.3.3 Dynamische thermische Gebäudesimulation

Um detaillierte Informationen über das dynamische Verhalten eines ganzen Gebäudes oder einzelner Räume zu erhalten, müssen instationäre Berechnungsverfahren angewendet werden. Der Aufwand (Modellbildung und Berechnung) bei der Anwendung von dynamischen Verfahren ist aber relativ hoch. Daher werden Energiekennzahlen nur in Ausnahmefällen mittels dynamischer Gebäudesimulation ermittelt, wenn diese in einem Projekt als Mittel zum Zweck einer hochwertigen Planung ohnehin vorgesehen ist.

2.1.4 Zukunftsaussichten

> Mit steigender Aktualität aller Energiefragen wächst auch das Interesse an der Möglichkeit, Voraussagen über Komfort und Energie im Gebäudeinneren machen zu können. [...]
>
> In Deutschland wurden in den letzten Jahren immer komplexere „Handberechnungsmethoden“ entwickelt, welche nun in der DIN-Norm 18599 verankert sind. Diese Norm umfasst rund 1.000 Seiten und ist in 10 Teile geteilt. Niemand wird auf die Idee kommen, diese Methoden auch wirklich „von Hand“, d.h. ohne gute Rechenwerkzeuge zu verwenden. [...]
>
> Handrechenmethoden schätzen die gesuchten Grössen mit zeitlichen Mittelwerten (gewöhnlich Monatswerten) ab. Der mögliche Einbezug verschiedenster Gebäudegeometrien sowie Anlagenkomponenten und deren Kombinationen führen zu derart komplizierten Algorithmen, dass diese kaum jemand freiwillig anwenden wird. Ausserdem ist es gegen die Natur der Sache, mit diesen Methoden die dank heutigem Wissen immer häufiger und geschickter genutzten dynamischen Vorgänge im Gebäude zu berücksichtigen. Entsprechend können Resultate aus monatsweisen Berechnungen im Gegensatz zu denjenigen aus dynamischen Simulationen auch nicht mit Messresultaten übereinstimmen. Die Monatsschrittweite müsste dazu so weit verkürzt werden, bis man quasi bei der dynamischen Simulation ankommt. Für eine dynamische Simulation wird kaum mehr Eingabeinformation benötigt als für eine monatliche Berechnung. [...]
>
> Fazit: Um dem steigenden Bedarf gerecht zu werden, werden Handrechenmethoden heute immer raffinierter, damit aber auch komplexer. Die in der Praxis sich durchsetzenden dynamischen Simulationsprogramme werden immer effizienter und sicherer bedienbar. Sie könnten deshalb schon bald selbst den täglichen Einsatz von Handrechenmethoden ersetzen. [Moo08]

Der wahrscheinlich wesentlichste Aspekt für die praktische Anwendbarkeit simulativer Berechnungsmethoden ist jener des Modellierungsaufwandes: Nur wenn es im wirtschaftlichen Rahmen möglich ist, Gebäudemodelle mit entsprechendem Detaillierungsgrad zu erstellen, so wird diese Methode in großem Stil Einzug in die Planung finden. In diesem Zusammenhang ist ein Blick auf aktuelle Entwicklungen im CAD-Sektor interessant:

Der Output aus *herkömmlichen* CAD-Architekturprogrammen ist inhaltlich nicht viel mehr als ein Plan, der auf klassische Weise zeichnerisch zu Papier gebracht wurde: eine Summe von Linien und Textzeichen. Erst in der Interpretation durch den Betrachter entsteht ein (imaginäres) Gebäudemodell, das sich aus Informationen aus dem Plan und Informationen/Erfahrungen, die dem Betrachter sonst noch zur Verfügung stehen zusammen setzt. Ein *ganzheitliches* Gebäudemodell entsteht im gesamten Planungsverlauf *nie*.

Aktuell ist aber eine neue Generation von CAD-Programmen für die Architektur im Aufkommen, die im Gegensatz zum herkömmlichen CAD auf einem objektorientierten Designansatz basiert. Mit CAD-Programmen der neuesten Generation werden aus virtuellen Bauteilen komplette, 3D-Gebäudemodelle erzeugt. Diesen virtuellen Bauteilen werden dabei nicht nur geometrische Eigenschaften zugewiesen, sondern auch sämtliche andere, im Planungsprozess relevanten Parameter (Materialkennwerte, Kosten, Herstellerinformationen, etc.).

Ein weiterer, wesentlicher Fortschritt in der neuesten Generation von CAD-Tools besteht in der Möglichkeit zum kollaborativen Arbeiten mehrerer Planer in einem Gebäudemodell (Computer Supported Collaborative Work – CSCW). Es ist leicht absehbar, dass diese Form des CAD früher oder später das herkömmliche CAD komplett ablösen wird.

Durch diese Form des CAD entstehen für alle Planungsbeteiligten Vorteile. In der thermisch-energetischen Planung besteht ein wesentlicher Teil des Aufwandes in der Eingabe von repräsentativen Gebäudemodellen (Geometrie, Bauteilkennwerte, Gebäudetechnik). Da ganzheitliche Gebäudemodelle, wie sie in modernen CAD-Programmen entstehen, bereits einen Großteil der nötigen Datenbasis für die Berechnung von Energiekennzahlen bzw. für Simulationen beinhalten, ist es nahe liegend, dass man sich diesen Umstand in Zukunft zunutze machen wird. Es ist also zu erwarten, dass dadurch Werkzeuge wie dynamische thermische Gebäudesimulation oder Strömungsberechnung mittels CFD verstärkt Einzug in den Planungsprozess finden werden.

Wegen der detaillierteren Ergebnisse, die eine Simulation liefern kann, ist anzunehmen, dass es in Zukunft auch standardisierte, simulationsähnliche Verfahren geben wird, mit deren Hilfe Energiekennzahlen ermittelt werden können. Trotzdem ist in der thermisch-energetischen Planung (ebenso wie auch in anderen Planungsgebieten des Bauwesens) auszuschließen, dass vereinfachte Rechenverfahren gänzlich durch komplexe numerische Simulationsverfahren verdrängt werden. Zu den „zukunftssicheren“ Anwendungsgebieten vereinfachter Rechenverfahren zählen z.B.:

- Einfache Probleme, bei denen der Detaillierungsgrad einer vereinfachten Betrachtung zur Problemlösung ausreicht. (Wirtschaftlichkeitsprobleme)
- Untersuchungsgebiete, die so groß sind, dass der Detaillierungsgrad einer numerischen Simulation nur punktuell erreicht werden kann. (Rechenkapazitätsprobleme)
- Optimierungen mittels Parametervariation, die so umfangreich sind, dass nur vereinfachte Berechnungsverfahren infrage kommen. (Rechenkapazitätsprobleme)
- Plausibilisierung von Ergebnissen aus „Simulations-Black-Boxen“ (Redundanzprobleme)

2.1.5 Warum Kopplung von vereinfachten und simulativen Methoden?

Es stellt sich als schwierig heraus, vereinfachte Rechenverfahren zur thermisch-energetischen Planung anhand realer Gebäude bzw. Testaufbauten zu validieren bzw. zu optimieren. Dies begründet sich unter anderem mit

- der Vielzahl an Eingangsparametern,
- der Komplexität und der Interaktion verschiedener Teilsysteme,
- der schwierigen Erfassung des tatsächlichen Nutzerverhaltens,
- der Abbildung von Steuerung bzw. Regelung gebäudetechnischer Anlagen, sowie
- dem schwierigen Erfassen von Abweichungen zwischen Planung und Bestand.

Die einzelnen Teilmodelle wie HWB und KB sind anhand von Testaufbauten noch überprüfbar, aber bei komplexeren (Gesamt-)Systemen werden „in situ“-Validierungen zunehmend schwieriger, und es wird zunehmend schwieriger verschiedene Parameterkombinationen experimentell zu überprüfen. Es ist also ein alternatives Werkzeug

gefragt, mit dessen Hilfe vereinfachte Rechenverfahren mit vertretbarem Aufwand evaluiert, validiert bzw. optimiert werden können. Dafür bietet sich die Computersimulation an.

Von hohem praktischem Nutzen wäre ein Werkzeug, das auf Basis *ein und des selben* digitalen Gebäudemodells sowohl eine vereinfachte Berechnung, wie auch eine detaillierte dynamische Simulation durchführen, und die (Zwischen-)Ergebnisse gegenüberstellen kann. Idealerweise soll es dabei möglich sein, einzelne Teilmodelle „gegeneinander auszutauschen“. Eine solche Kombination aus diskreten Zeitschrittverfahren (wie etwa dem Monatsbilanzverfahren) und kontinuierlichen Simulationsmodellen kann mithilfe eines hybriden Simulationsansatzes gelöst werden. Mit Modelica steht eine Umgebung zur Verfügung, in der sich derartige Lösungen umsetzen lassen.

2.2 Modelica

Modelica ist eine objektorientierte Modellbildungssprache zur Darstellung komplexer physikalischer Systeme. Die Sprache wurde Anfang der 1990er-Jahre „erfunden“ und wird seither laufend weiterentwickelt. Derzeit ist Version 3.2 [Mod10] der Sprachspezifikation aktuell.

Modelica ist ein offener, nicht proprietärer, implementationsneutraler Standard. Dadurch eröffnen sich für Drittparteien und Benutzer vielfältige Möglichkeiten zur Entwicklung eigener Modellbibliotheken für beliebige physikalische Gebiete. Modelica sowie einzelne Bibliotheken und Tools sind frei verfügbar.

2.2.1 Vor- und Nachteile von Modelica

Gegenüber anderen Modellbildungs- und Simulationssprachen weist Modelica folgende Vorteile auf:

- ⊕ Es gibt eine standardisierte Sprachstruktur, welche die „üblichen“ Vorteile einer *objektorientierten Programmiersprache* (unter anderem: Objekthierarchie, Mehrfachvererbung, Zugriffskontrolle, Templates, Interfaces, etc.) in einer einzigen Klassenstruktur vereinigt. Dadurch wird die Wiederverwendung von Modellen stark vereinfacht und der Austausch zwischen verschiedenen Anwendern wesentlich erleichtert.

⊕ Die *Modellbildung erfolgt akausal* auf Basis differential-algebraischer Gleichungssysteme (DAEs). Es werden also Differentialgleichungen und algebraische Gleichungen aufgestellt, um ein System physikalisch zu beschreiben. Mit anderen Worten erfolgt durch mathematische Beschreibung der physikalischen Zusammenhänge eine vollständige Abbildung des Systems.

⊕ Modelica bietet auch Mittel zur kausalen Modellbildung und zur *Integration von kausalen und akausalen Teilmodellen* in einem System. Dadurch lassen sich z.B. Steuerungs- und Regelungssysteme im „klassischen“ Blockdiagramm-Stil abbilden, während die angesteuerten Anlagen im selben System realitätsnah als akausal modellierte Einheiten abgebildet werden können.

⊕ Es besteht die Möglichkeit zur *Multi-Domain-Modellierung*, d.h., thermische, mechanische, hydraulische, elektrische oder pneumatische Komponenten können gemeinsam modelliert und simuliert werden.

Die Nachteile von Modelica als Programmiersprache zur praktischen Anwendung bestehen nach Ansicht des Autors darin, dass

⊖ nur *unvollständige Implementationen* der Sprache entsprechend der Modelica Language Specifications 3.x [Mod07] [Mod09] [Mod10] existieren, sowie darin, dass

⊖ einige Modelica-Umgebungen den üblichen Anforderungen an *produktionsstabile Software* nicht gerecht werden.[1]

2.2.1.1 Modelica vs. Tabellenkalkulation

Bei Verwendung von Tabellenkalkulationsprogrammen stößt der Anwender schon an deren Grenzen, noch bevor es um Simulation im eigentlichen Sinn geht. Selbst Berechnung von Energiekennzahlen nach vereinfachten Verfahren gemäß der in ➪ Abschnitt

1 Im Zuge der Ausarbeitung des gegenständlichen Projekts wurde ein wesentlicher Teil der Zeit damit verbracht, sich (gezwungenermaßen) mit diesen Problemen auseinander zu setzen. Um zukünftigen Anwendern bzw. Bearbeitern der Programmierung das Leben etwas leichter zu machen, werden die wesentlichen Stolpersteine in ➪ Abschnitt A beschrieben.

2.1.3.1 erwähnten Normen gibt es schon „spürbare“ Rechnenzeiten und Übersichtlichkeitsprobleme. Beim ersten Schritt weg vom Status quo der Gebäudeenergieausweis-Berechnung in Richtung einer detaillierten Betrachtung (Multi-Zonen-Modelle, dichteres Zeitgitter, instationäre Berechnungen, o.Ä.) wird die Grenze zur Praxis-Untauglichkeit von Tabellenkalkulationsprogrammen überschritten.

2.2.1.2 Modelica vs. Low-Level-Programmiersprachen (C, C++, Fortran)

Sowohl *MathModelica* und *OpenModelica*, wie auch *Dymola* stützen sich im Endeffekt auf C++ als Low-Level-Programmiersprache, d.h. die jeweiligen Compiler-Backends generieren C++-Code, der dann im letzten Schritt mithilfe eines C-Compiliers in ein systemspezifisches, ausführbares Simulationsprogramm übersetzt wird (➪ siehe Abbildung 2.5).

Somit sind Modelica-Umgebungen in der Regel Systeme, welche es dem Programmierer ermöglichen, sich mittels einer spezialisierten High-Level-Programmiersprache – eben Modelica – auf die Formulierung des physikalischen Systems zu konzentrieren, ohne dass dabei die Performanzvorteile einer systemnahen (Low-Level) C++-Programmierung verloren gehen. Der Programmierer wird aber von der Implementation der Komponenten zur numerischen Lösung des zugrunde liegenden DAE Systems (Analyzer, Optimizer, Solver) entlastet. Außerdem stehen fertig optimierte Funktionen zur Formulierung von Differentialbeziehungen sowie für Vektor- und Matrixoperationen zur Verfügung.

Die Syntax von Modelica erinnert im Übrigen an jenen von Fortran, wo ebenfalls eine leicht lesbare, mathematische Beschreibung im Vordergrund steht – ein Vorteil, der bei C++nicht besteht. Ein weiterer angenehmer Aspekt bei der Programmierung bzw. Modellierung mit Modelica besteht darin, dass das Modell und dessen Teilmodelle jederzeit auf mathematische Lösbarkeit (Über-/Unterdeterminiertheit) überprüft werden können. Außerdem kann sich der Modelica-Anwender zur Auswertung der generierten Simulationsergebnisse auf bestehende Auswertungs-Tools stützen.

☞ **Hinweis:** In Einzelfällen – z.B. bei der Verwendung sehr großer Datentabellen in einer Simulation – kann es nötig sein, dass sog. externe Funktionen bzw. externe Datenobjekte entsprechend [Mod07] Kap. 12.8 zur Anwendung kommen. Diese Konstrukte müssen dann in C oder Fortran programmiert werden, d.h. zum Teil kommt man auch als Modelica-Entwickler nicht um die Programmierung in

Low-Level-Programmiersprachen herum. Im konkreten Fall der entwickelten Simulationsumgebung war es z.B. nötig, die Datentabellen von Testreferenzjahren – das sind 8760 x 15 Elemente umfassende Matrizen – als externe Objekte zu implementieren, welche über externe Funktionen angesprochen werden.

2.2.1.3 Modelica vs. Matlab/Simulink

Der zentrale Unterschied zwischen Modelica und Matlab/Simulink besteht sicher in der Art der Abbildung der Systeme: Während ein System bei der akausalen Modellierung (Modelica) über seine physikalisch-mathematischen Zusammenhänge *als Ganzes* beschrieben wird, werden beim kausalen Ansatz (Matlab/Simulink) einzelne Rechenschritte in einer feststehenden *Ablaufkette* definiert[1].

Es ist in diesem Zusammenhang wichtig zu verstehen, dass zwar in beiden Umgebungen eine grafische Systemdarstellung mittels Blockdiagrammen möglich ist, dass aber (physikalische Modellierung in Modelica vorausgesetzt) völlig unterschiedliche Konzepte dahinter stecken, die auch unterschiedlich zu interpretieren sind: So stellt eine Verbindungslinie zwischen graphischen Symbolen in Modelica-Diagrammen immer eine *Gleichung im physikalischen Modell* dar, und das Modell als Ganzes repräsentiert ein *Gleichungssystem*. In Simulink stellt eine solche Linie üblicherweise die *Übergabe eines Wertes* von einem Block an den nächsten dar, und ein Modell ist eine *Ablaufkette aus einzelnen Rechenschritten*.

Eine umfassende und übersichtliche Gegenüberstellung beider Ansätze wird z.B. in [Hom06], Kap. 2.4 ff. gegeben:

> Die kausale Modellierungsmethodik lässt sich durch folgende Eigenschaften charakterisieren:
>
> - Das zu beschreibende System wird in regelungstechnische Standardblöcke zerlegt, die das Systemverhalten der Regelstrecke erkennen lassen.
> - Das Blockdiagramm entspricht einer Reihe von Anweisungen, die die Zustandsableitungen berechnen.

1 Es ist zu erwähnen, dass die Grenzen zwischen unterschiedlichen Modellbildungs-Ansätzen zunehmend unschärfer werden. In jüngerer Vergangenheit wurden Schnittstellen zwischen *Dymola* und *Simulink* bzw. zwischen *MathModelica* und *Mathematica* entwickelt. Auch Modelica als Programmiersprache weist alle Elemente auf, die für kausale Systembeschreibung nötig sind.

- Die Ordnung des Systems und die Kopplung der Zustände sind transparent.

Die blockorientierte Modellierung hat folgende Nachteile:

- Die Einsicht in die physikalische Struktur geht verloren.
- Die Blockdiagramme bzw. Strukturbilder sind schwer lesbar.
- Die Austauschbarkeit und damit Wiederverwendbarkeit von Teilmodellen wird erschwert.
- Eine kleine Änderung der physikalischen Struktur bedingt unter Umständen ein völliges Neuaufstellen des Blockdiagramms.
- Die Modellierung strukturvariabler, nichtlinearer Systeme ist entsprechend aufwendig und unübersichtlich (Diode, Kupplung, mechanischer Freilauf) und teilweise unmöglich.

Die akausale Modellierungsmethodik läßt sich durch folgende Eigenschaften charakterisieren:

- Das zu beschreibende System wird durch gekoppelte Standardbauteile, die gewöhnlich in Bibliotheken verfügbar sind, dargestellt.
- Die physikalische Struktur bleibt vollständig erhalten.
- Die Beschreibung entspricht den lokalen physikalischen Gleichungen der Bauteile, die von ihrer Umgebung unabhängig sind, sowie deren Kopplung zu einem Gesamtgleichungssystem.
- Das differential algebraische Gleichungssystem wird durch symbolische Transformation in eine für den Solver lösbare Form umgewandelt und numerisch integriert. Die physikalisch orientierte Modellierung hat folgende

Nachteile:

- Die regelungstechnische Streckenstruktur ist nicht erkennbar.

Aus Modellierungssicht gibt es folgende Vorteile des akausalen Ansatzes: Da bei der akausalen Modellierung nicht signalorientiert modelliert wird, kann physikalisch strukturiert werden – unabhängig vom kausalen Zusammenhang der Zustandsgrößen. Teilmodelle können da voneinander getrennt werden, wo es natürlich sinnvoll erscheint. Außerdem wird die Austauschbarkeit und Wiederverwendbarkeit von Komponenten und damit das verteilte Modellieren gewährleistet und die hierarchische Strukturierung des Modells unterstützt. Globale algorithmische Fallunterscheidungen sind bei strukturvariablen Systemen nicht notwendig. Das Modell ist in seiner physikalischen Struktur leicht änderbar. Es lassen sich physikalische Sachverhalte in Form von DAEs beschreiben, die sich häufig nur

mit großem (händischen) Aufwand oder gar nicht in ein System von ODEs überführen lassen.

Aus regelungstechnischer Sicht scheint ein Vorteil des kausalen Ansatzes zu sein, daß die systemtheoretische Struktur der Regelstrecke erkennbar ist und damit die Ordnung des Systems. Die Kopplungen der Zustände untereinander sind transparent. Der kausale Ansatz eignet sich eher für die algorithmische Beschreibung von Systemen, da Signalflüsse modelliert werden. ([Hom06], Kap. 2.4 ff.)

2.2.2 OpenModelica

OpenModelica ist eine unter OSMC Public License [Lin10] lizenzierte Modelica-Umgebung. Diese Lizenz bietet dem Anwender mehrere Möglichkeiten zur Lizenzierung der Software. Eine dieser Möglichkeiten ist die Lizenzierung unter GPL v3 [Fre07], d.h. dem Lizenznehmer steht auch der Quelltext der Software zur Verfügung und es besteht die Möglichkeit, beliebige Änderungen am *OpenModelica* System vorzunehmen.

OpenModelica wird vom Open Source Modelica Consortium (kurz OMSC; ➪ `http://www.openmodelica.org/index.php/home/consortium`) gepflegt und kostenlos zur Verfügung gestellt. Die Hauptentwickler von *OpenModelica* arbeiten großteils am ➪ Department of Computer and Information Science der ➪ Universität Linköping, Schweden.

Mit dem Modelica Development Tooling (MDT) - Plugin ist es möglich, das weit verbreitete Eclipse IDE (➪ `http://www.eclipse.org`) zur Softwareentwicklung mit Modelica zu nutzen. Speziell große Projekte profitieren stark von den Möglichkeiten, die sich durch ein professionelles IDE eröffnen, und die in keiner der kommerziellen Modelica-Versionen auch nur annähernd zur Verfügung stehen:

- Multiple-Document-Interface (MDI) zur gleichzeitigen Bearbeitung mehrerer Dateien
- Vielfältige Möglichkeiten zur Darstellung und Navigation in großen Projekthierarchien
- Team- und Versionsmanagement (CVS, SVN, GIT,...)

- Syntax-Highlighting, Syntax-Prüfung und Markierung von Syntax-Fehlern während der Eingabe
- Auto-Vervollständigung und Auto-Lookup von Bezeichnern im Sourcecode
- Leistungsfähige Such- und Ersetz-Funktionen (u.a. mit regulären Ausdrücken)
- Bookmarks & To-Do-Listen

Vor dem Hintergrund, dass das gegenständliche Projekt relativ umfangreich ist und mehrere Entwickler daran beteiligt sind, war es undenkbar, bei der Entwicklung auf diese Vorteile zu verzichten.

Die offensichtlichen Nachteile der *OpenModelica*-Umgebung sind:

- Fehlende Möglichkeit zur graphischen Modellierung („Blockdiagramme“)
- Unhandliche Simulationsumgebung, wenn es um Parametervariation o.Ä. geht
- Nur sehr rudimentäre Auswertungsmöglichkeiten von Simulationsergebnissen

Solange es um Modelica-Programmierung geht, kann *OpenModelica* also mit der (nach Meinung des Autors) konkurrenzlosen Entwicklungsumgebung (MDT) punkten. Sobald es aber um graphische Modellierung und Simulations-Auswertung geht, kann *OpenModelica* nicht mehr mit den kommerziell verfügbaren Modelica-Umgebungen konkurrieren. Für diese Zwecke muss auf eine kommerzielle Modelica-Umgebung zurückgegriffen werden.

Der wirkliche Haken an *OpenModelica* im Zusammenhang mit dem gegenständlichen Projekt war aber, dass sich der *OpenModelica*-Compiler (OMC) zur Zeit der Entwicklung gerade mitten in einem großen Versionssprung (von der veralteten Version 1.4.5 auf 1.5.0) befunden hat und deshalb alles andere als andere als fehlerfrei war (➪ siehe Abschnitt A.2.1).

Da *OpenModelica* kein kommerzielles Produkt ist, gibt es auch keinen Support im eigentlichen Sinn. Es existieren aber ein Internet-Forum (➪ `http://www.openmodelica.org/index.php/forum`) sowie die Mailing-List *OpenModelicaInterest*[1] (OpenModelicaInterest@ida.liu.se), über die sich die *OpenModelica*-Community austauscht. Viele der Anfragen werden von den Entwicklern von *OpenModelica* selbst beantwortet,

1 Ein Archiv der Mailing-List ist online unter ➪ `http://www.mail-archive.com/openmodelicainterest@ida.liu.se/` verfügbar.

wobei einfache Fragen häufig innerhalb weniger Stunden beantwortet werden, während komplexe Fragen manchmal etwas länger dauern, oder – falls sie nicht ausreichend genau gestellt werden – überhaupt „liegen bleiben“. Insgesamt funktioniert der freiwillige Support durch die Community aber eigentlich besser als jener der kommerziellen Anbieter von Modelica-Umgebungen.

2.2.3 Kommerzielle Modelica-Umgebungen

2.2.3.1 MathModelica

MathModelica ist ein Produkt der Firma *MathCore Engineering AB* mit Sitz in Linköping, Schweden. Dies ist insofern interessant, als auch *OpenModelica* (➪ siehe Abschnitt 2.2.2) und der Modelica-Sprachstandard (➪ siehe Abschnitt 2.2) in Linköping ihren Ursprung haben. Bei der anfänglichen Evaluierung verschiedener Modelica-Umgebungen sind dem Autor schon „verdächtige“ Parallelen in den Outputs des *OpenModelica*-Compilers und jenen des *MathModelica*-Kernels aufgefallen, auf denen sich schnell die Vermutung gegründet hat, dass es sich beim Unternehmen *MathCore* um ein Research Spin-Off der Universität Linköping handelt. Im Zuge einer Support-Anfrage an *MathCore* hat sich diese Vermutung später bestätigt:

> Yes you are correct to assume that MM is „kind of commercial spin-off“ with emphasis on the „kind of“. The two projects have been running in parallel for several years and MathModelica is in fact using part of the OM kernel, namely the part („kernel front-end“) that is flattening the Modelica code (basically setting up the system of equations as one large, flat system). For the rest of the process OM and MM use different („kernel back-ends“). The back-end takes the flattened set of equations, prepares and optimizes it for simulation, and then simulates it. At the moment the Open Source Modelica Consortium is focusing on completing the flattening of the Modelica language, while MathCore (MathModelica) is focusing on the back-end. (Email *MathCore* 09.08.2010 09:15)

Konkret benutzt der *MathModelica*-Kernel das *OpenModelica*-Compiler-Frontend – also jenen Teil von *OpenModelica*, der den Modelica-Code für die Übersetzung in C++-Code vorbereitet. Wie auch *OpenModelica*, stützt sich *MathModelica* per default auf GCC als C++Compiler, wobei optional auch der *Microsoft Visual C++* Compiler eingesetzt werden kann. Das Programm verfügt über eine Schnittstelle zur Mathematik-Software *Mathematica* (Fa. *Wolfram Research*, USA).

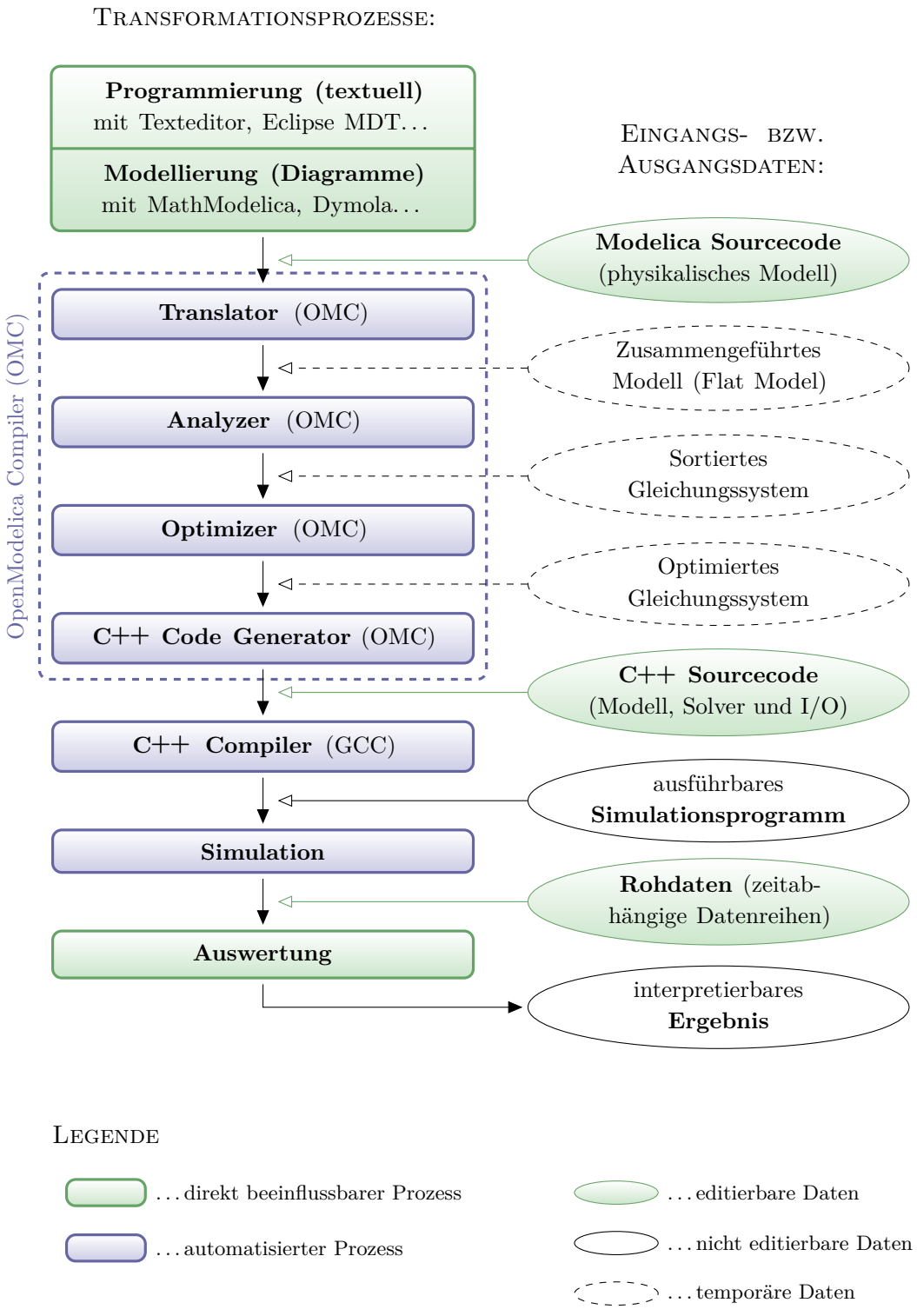


Abbildung 2.5: Typisches Ablaufschema vom Modelica-Quelltext zum Simulationsergebnis am Beispiel von *OpenModelica*

Der Texteditor von *MathModelica* ist eher rudimentär und konzentriert sich auf gute Lesbarkeit von Modelica-Code (Syntax-Highlighting und ab Version 3.0 auch Annotation-Canceling). Zum Schreiben von Code ist der Editor nur bedingt geeignet, und *MathModelica* kann relativ leicht zum Absturz gebracht werden, wenn man im Editor Änderungen direkt am Code vornimmt.

Dass *MathModelica* aus dem Umfeld der *OpenModelica*-Entwickler stammt, welche ihrerseits wiederum die Entwickler des Modelica Sprachstandards sind, ist sicher ein potenzieller Vorteil dieser Modelica-Umgebung. Darüber hinaus macht das GUI von *MathModelica* nach Ansicht des Autors ein deutlich besseres Bild, als jenes von *Dymola*. So sind in *MathModelica* Modellierung und Simulation & Auswertung in separaten (aber gut zusammenspielenden) Applikationen umgesetzt. *MathModelica* verfügt außerdem über die (nach Ansicht des Autors) beste Rendering-Engine zur graphischen Darstellung von Symbolen und Modellen.

Diese „Verwandtschaft“ mit *OpenModelica* – und hier insbes. die potenziell beste Kompatibilität mit der leistungsfähigen MDT Entwicklungsumgebung – sowie das fortschrittlichere GUI haben den Autor zu Beginn des Projektes dazu veranlasst, auf *MathModelica* als Tool für Modellierung, Simulation und Auswertung zu sezten. Theoretisch sind also *OpenModelica*-MDT für die Programmierung und *MathModelica* für Modellierung, Simulation & Auswertung eine Kombination mit „maximaler Durchschlagkraft“. Aber...

Es ist wohl kein Zufall, dass fast zeitgleich mit dem Versionssprung von *OpenModelica* auch ein Versionssprung von *MathModelica* (von Version 2.1 auf 3.0) einher gegangen ist, und der geschätzte Leser dieser Diplomarbeit erahnt an dieser Stelle wahrscheinlich schon, welche Konsequenzen dieser Umstand hatte: Das *MathModelica*-Frontend hatte mehrere Fehler aus dem Versionsprung im *OpenModelica*-Frontend „geerbt“, und mit einigen Problemen im Zusammenhang mit eigenen Versionssprung „ergänzt“ (➪ siehe Abschnitt A).

Kurz: Es war trotz intensivster Bemühungen nicht möglich, die geplante Architektur der entwickelten Simulationsumgebung in *MathModelica* vollständig lauffähig zu machen, und man musste sich schließlich auf *Dymola* als Tool für Modellierung, Simulation & Auswertung zurück ziehen.

Positiv hervorzuheben ist, dass bei *MathCore* immerhin der Support sehr gut funktioniert. Die Antwortzeiten liegen im Bereich von wenigen Tagen und sind damit um

einiges kürzer als dies bei anderen Anbietern der Fall ist. Außerdem hat der Autor beim Umstieg auf *Dymola* auch einige sehr praktische Features von *MathModelica* bitter vermisst: So gibt es in *Dymola* etwa keine „Load & Replace"-Funktion, mit der bereits geladene Bibliotheken ohne Neustart des Programms neu geladen werden können – ein Feature, das besonders nützlich ist, wenn die Bibliotheken wie im gegenständlichen Fall in einem externen Programm (z.B. Eclipse MDT) programmiert werden.

2.2.3.2 Dymola

Dymola ist ein Produkt der Firma *Dassault Systemes AB* mit Sitz in Lund, Schweden. Das Tool stützt sich per default auf den *Microsoft Visual C++* Compiler und verfügt über ein Interface zu *Matlab/Simulink* der Firma *MathWorks*. Im Gegensatz zu *MathModelica* sind alle Komponenten der Software (Modellierung, Simulation & Auswertung) in einer einzigen Applikation vereint.

Der Texteditor von *Dymola* konzentriert sich ähnlich wie jener von *MathModelica* auf gute Lesbarkeit von Modelica-Code (Syntax-Highlighting und Annotation-Canceling) und ist zum Schreiben von Code nur bedingt geeignet.

Als einzige der untersuchten Modelica-Umgebungen führt *Dymola* eine Prüfung der verwendeten physikalischen Einheiten durch. Beim Umstieg von *MathModelica* auf *Dymola* sind Kompatiblitätsprobleme hauptsächlich im Bereich der Interpretation von graphischen Annotationen aufgetreten, wo *Dymola* einige Schwächen aufweist. Im *Dymola*-Kernel gibt es dagegen nur wenige Fehler, die es zu umgehen gilt (➪ siehe Abschnitt A). Dass seit dem Umstieg auf *Dymola* die Dinge plötzlich wesentlich „runder" laufen, hat aber vor allem zwei Gründe:

1. Zwar ist auch der *Dymola*-Kernel nicht fehlerfrei, aber die meisten Modelica 3.0 Features sind *stabil* implementiert, sodass man (im Gegensatz zu *OpenModelica* und *MathModelica*) als Entwickler beim Auftreten eines Fehlers normalerweise zuerst davon ausgehen soll, dass man diesen selbst verursacht hat.

2. Der *Dymola*-Kernel liefert bei Fehlern in der Programmierung wesentlich besser interpretierbare Error-Meldungen, die es gegenüber den anderen untersuchten Modelica-Umgebungen deutlich einfacher machen, Probleme im Sourcecode einzugrenzen.

Wenn man als Entwickler trotzdem einmal nicht weiterkommt, so sollte man sich allerdings nicht auf den Support bei *Dassault Systemes* verlassen. Die Antwortzeiten auf Support-Anfragen liegen teilweise im Bereich von über einer Woche, und es ist im Zuge des Projektes auch vorgekommen, dass Support-Anfragen nach über einer Woche Wartezeit noch immer nicht beantwortet waren, sondern erst nach erneutem Nachfragen bearbeitet wurden.

2.2.4 Zusammenfassung

Nach Einschätzung des Autors ist der einzige derzeit wirklich produktionsstabile Modelica-3.x-Kernel jener von *Dymola*. Der *MathModelica*-Kernel weist derzeit einige Probleme auf, die zwar bald behoben sein könnten, aber momentan einen Einsatz der Software in größeren Entwicklungsprojekten erheblich erschweren (➪ siehe Abschnitt A). Die Fehler im *OpenModelica*-compiler (OMC) sind aktuell so schwerwiegend, dass sie die Verwendung von *OpenModelica* in Modelica-3.x Entwicklungsprojekten de facto verhindern (➪ siehe Abschnitt A.2.1). Für die Verwendung als reiner Syntax-Parser für Eclipse MDT ist der OMC aber geeignet (und unerlässlich).

Mit Eclipse MDT stellt *OpenModelica* eine integrierte Entwicklungsumgebung zur Verfügung, die kaum Wünsche offen lässt, und die momentan nur unter der Instabilität des im Hintergund arbeitenden OMC leidet, weil dieser gelegentlich abstürzt. Keine der kommerziellen Modelica-Umgebungen bietet eine vergleichbare Umgebung mit Features für professionelle Softwareentwicklung, wie in ➪ Abschnitt 2.2.2 beschrieben. Die in den kommerziellen Paketen integrierten Editoren beschränken sich auf Features zur leichteren Lesbarkeit von Modelica-Code (Syntax-Highlighting und Annotation-Canceling).

MathModelica bietet von allen untersuchten Modelica-Umgebungen das am besten durchdachte GUI zur Modellierung, Simulation und Auswertung von Simulationsergebnissen.

Derzeit hat man mit den drei untersuchten Tools also

- eines, das einen stabilen, 3.x-konformen Modelica-Kernel zur Verfügung stellt (*Dymola*),
- eines, das ein leistungsfähiges IDE zur Programmierung bietet (Eclipse MDT von *OpenModelica*), und

- eines, das mit einem sehr gut durchdachten GUI die Modellierung, Simulation und Auswertung erleichtert (*MathModelica*).

Wenn in jenen Modelica-Umgebungen, die derzeit einen größeren Versionssprung durchmachen bzw. unmittelbar hinter sich haben, die damit verbundenen Fehler bereinigt sind, kann sich diese Situation zwar wieder etwas verschieben, aber in absehbarer Zukunft wird es wohl kein Modelica-Tool geben, das allen Anforderungen gerecht wird.

3 Umsetzung

3.1 Grundsätzliches

3.1.1 Kombination unterschiedlicher Zeitschrittweiten: Hybride Zeitgitter

Ziel ist es, typische Norm-Rechnungsmethoden (insbes. jene der in ➪ Abschnitt 2.1.3.1 aufgelisteten Normen) auf eine Art in Modelica abzubilden, die es erlaubt, solche vereinfachten Rechenmethoden mit simulationstypischen Methoden zu koppeln. Daraus ergibt sich die Notwendigkeit eines „hybriden“ Zeitgitters mit 12 Monats- bzw. 8760 Stunden-Zeitschritten, wobei die im vereinfachten Verfahren untersuchten Größen typischerweise am Monats-Zeitgitter liegen und die simulierten Größen am Stunden-Zeitgitter.

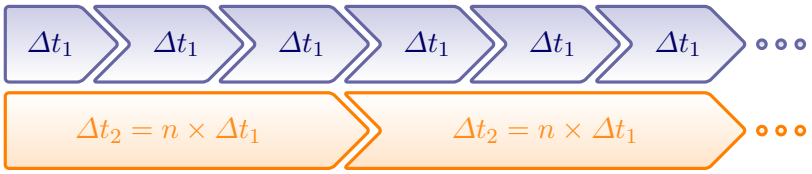


Abbildung 3.1: Synchronisierbare Zeitgitter mit verschiedenen Schrittweiten

Für jedes Zeitgitter einen unabhängigen Simulationsprozess zu erstellen, würde eine enge Kopplung beider Rechenmethoden deutlich erschweren, da für eine Kopplung mehrerer getrennter Prozesse ein hoher Aufwand erforderlich ist, um die einzelnen Simulationsprozesse zu synchronisieren und zu koordinieren:

- Erstellung von Interfaces für den Datenaustausch zwischen beiden Prozessen
- Formulierung von Protokollen für den Datenaustausch zwischen Prozessen

- Schaffung von Interrupt-Mechanismen zur Synchronisation der einzelnen Simulationsprozesse
- Erstellung einer Middleware, die unter Verwendung der ersten drei aufgezählten Elemente die Koordination der einzelnen Simulationsprozesse übernimmt.

Vor dem Hintergrund, dass beide Berechnungsmethoden in der gleichen Software-Umgebung (Modelica) implementiert werden sollten, ist es naheliegend, beide Methoden in einem einzigen Rechenprozess synchron zu implementieren. Damit das gelingt, müssen die Zeitgitter insofern synchronisierbar sein, als das Ende jedes groben Zeitschrittes zeitgleich mit dem Ende eines feinen Zeitschrittes stattfindet. Mit anderen Worten: Jedes Grobintervall muss sich in eine *ganzzahlige* Anzahl von Feinintervallen gliedern (➪ siehe Abbildung 3.1).

So entsteht ein virtuelles Zeitgitter aus Grobintervallen (zu je einem Monat), die sich jeweils in Feinintervalle (zu je einer Stunde) unterteilen. „Virtuell“ deshalb, weil der eigentliche Simulationsprozess am Stunden-Zeitgitter läuft und das Monats-Zeitgitter innerhalb dieses Stunden-Zeitgitters erzeugt wird, indem bestimmte (diskrete) Teile der Simulation nur in Monats-Intervallen variieren – und zwar genau jene Teile, die in den diskreten, quasi-stationären Methoden des Monatsbilanzverfahrens (➪ siehe Abschnitt 2.1.3.1) zur Anwendung kommen. Je nach gegenseitiger Abhängigkeit von Größen am Monats- und am Stundenraster ergeben sich theoretisch folgende Fälle:

1. Eine Größe auf einem Zeitgitter ist ausschließlich abhängig von Größen am anderen Zeitgitter.
2. Eine Größe auf einem Zeitgitter ist teilweise abhängig von Größen am anderen Zeitgitter.
    a) Die Größe am groben Zeitgitter ist abhängig von Größen am feinen Zeitgitter (➪ siehe Abschnitt 3.1.1.1)
    b) Die Größe am feinen Zeitgitter ist abhängig von Größen am groben Zeitgitter (➪ siehe Abschnitt 3.1.1.2)
    c) Die Größe auf einem Zeitgitter ist abhängig von Größen am anderen Zeitgitter, wobei diese wiederum von anderen Größen auf ersterem Zeitgitter abhängig sind (Zirkelbeziehung zwischen beiden Zeitgittern). (➪ siehe Abschnitt 3.1.1.3)

3. Eine Größe auf einem Zeitgitter ist unabhängig von Größen am anderen Zeitgitter.

Fall 1 ist trivial: Die Größe liegt offensichtlich am falschen Zeitgitter. Fall 3 ist ebenfalls trivial: Es besteht kein Kopplungsbedarf. In Fall 2 sind zur Kopplung verschiedene Ansätze nötig:

3.1.1.1 Größen am groben Zeitgitter mit Abhängigkeit von Größen am feinen Zeitgitter

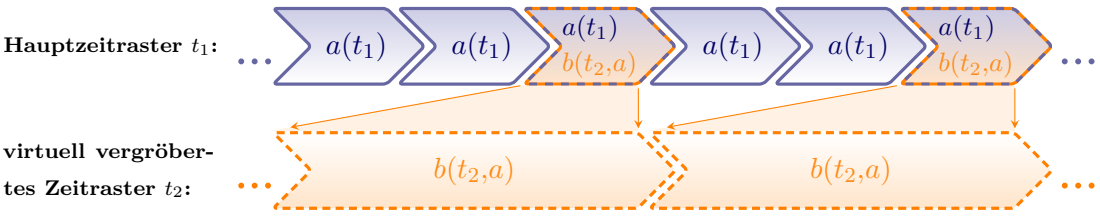


Abbildung 3.2: Hybride Zeitgitter ohne Rückkopplung (Rückwärts-Synchronisation)

Die Größe am groben Zeitgitter $b(t_2,a)$[1] wird immer in jenem Feinintervall ermittelt, das gemeinsam mit dem virtuellen Grobintervall *endet*, d.h., sie wird auf Basis der Ergebnisse für $a(t_1)$ aus den Feinintervallen *rückwirkend* fürs ganze Grobintervall berechnet:

1 Die gewählte Schreibweise mit t_1 und t_2 soll veranschaulichen, dass eine Größe $f(t)$ am feinen Zeitgitter ($f(t_1)$) oder am groben Zeitgitter ($f(t_2)$) liegt. Die Schreibweise soll nicht so missverstanden werden, dass t_1 und t_2 unterschiedliche Zeitpunkte bzw. Zeitlinien sind, sondern mit den Indizes 1 und 2 sollte veranschaulicht werden, mit welchen Zeitschrittweiten die jeweilige Größe ermittelt wird.

3.1.1.2 Größen am feinen Zeitgitter mit Abhängigkeit von Größen am groben Zeitgitter

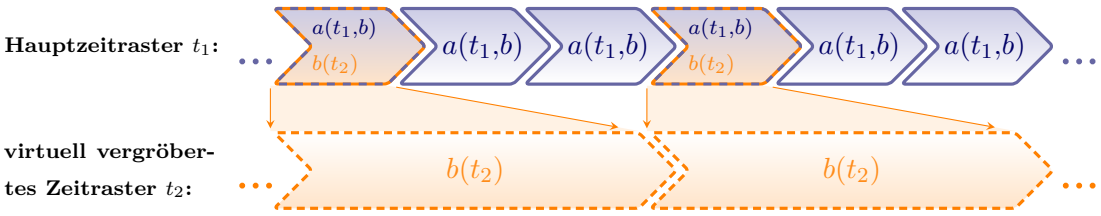


Abbildung 3.3: Hybride Zeitgitter ohne Rückkopplung (Vorwärts-Synchronisation)

Die Größe am feinen Zeitgitter $a(t_1,b)$ wird immer in jenem Feinintervall ermittelt, das gemeinsam mit dem virtuellen Grobintervall *beginnt*, d.h. sie wird auf Basis der „*voraus berechneten*“ Ergebnisse für $b(t_2)$ fürs jeweilige Feinintervall berechnet:

3.1.1.3 Zirkelbeziehungen zwischen Größen am groben und am feinen Zeitgitter

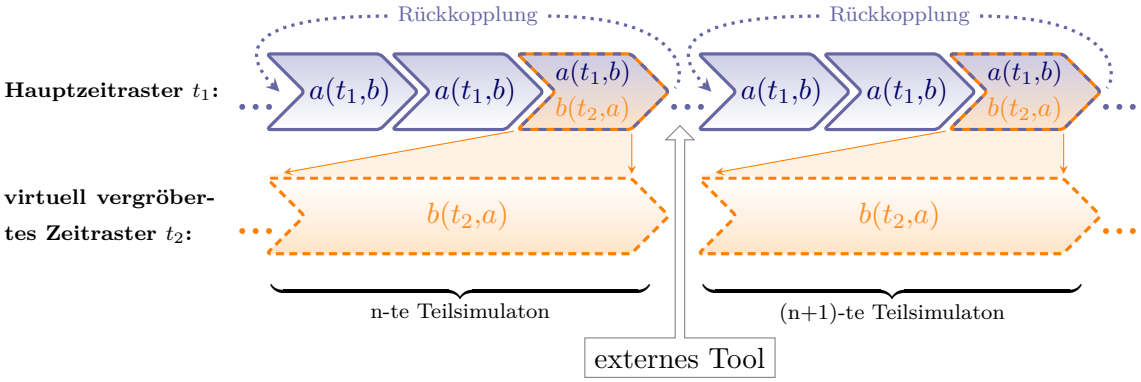


Abbildung 3.4: Hybrides Zeitgitter mit Rückkopplung (Zirkelbeziehung)

Im Prinzip wird dieses Verfahren gelegentlich bei gekoppelten Simulationen mit unterschiedlichen Simulationstools angewandt. Üblicherweise wird in solchen Fällen eine Middleware als Schnittstelle und Koordinationstool zwischen den Programmen

eingesetzt[1]. Weil die Iteration mit zunehmender Anzahl gekoppelter Teilsysteme sehr aufwändig wird, wird in solchen Fällen üblicherweise versucht, die Rückkopplung auf Modellebene aufzulösen.

Wenn – wie im gegenständlichen Projekt – eine Rückkopplung noch dazu auf das selbe Software-System erfolgt, ist es naheliegend, die Zirkelbeziehung aufzulösen, indem man

- entweder die rückkoppelnden Größen aufs andere Zeitgitter portiert, und so ein System wie in ➪ Abschnitt 3.1.1.1 oder ➪ Abschnitt 3.1.1.2 beschrieben erzeugt,
- oder überhaupt das gesamte betroffene Teilsystem auf *ein* Zeitgitter portiert.

Die Funktionen zur Erzeugung des virtuell vergröberten Zeitgitters in der erstellten Simulationsumgebung werden in ➪ Abschnitt 3.2.4.5 angeführt.

3.1.2 Kombination diskret-kontinuierlich, stationär-instationär bzw. kausal-akausal: Hybride Modelle

Die große Stärke von Modelica ist, dass sich praktisch alle in ➪ Abbildung 2.1 dargestellten Simulationsaspekte *nahtlos* kombinieren lassen. Die „Gefahr“ dieser Flexibilität ist, dass Modelle entstehen, deren Ergebnisse nicht mehr repräsentativ oder nicht mehr interpretierbar sind.

Dieses Potenzial besteht sicher auch im konkreten Fall einer Kombination von diskret-stationären „Handrechenmethoden“ mit dynamischen Teilsystemen, aber im Zuge dieser Arbeit wurden diesbezüglich noch keine Erfahrungen gesammelt. Einige Grundregeln für die hybride Modellierung können festgelegt werden:

- Physikalisch stark wechselwirkende Teilsysteme sollten auf möglichst gleiche Arten abgebildet werden. So würde es etwa wenig Sinn machen, die Lufttemperatur in einem Raum mit einem dynamischen Modell zu untersuchen, wenn die raumbegrenzenden Flächen nur stationär abgebildet sind.

1 Ein Beispiel, in dem Tool-Coupling mit Modelica als eine der Komponenten erfolgt ist, wird in [Kos06] beschrieben.

- Elemente gleichen oder ähnlichen Typs sollten gleich oder ähnlich repräsentiert sein. So würde es z.B. keinen Sinn machen, den Wärmestrom durch bestimmte Teile der thermischen Gebäudehülle – etwa durch die Wände – dynamisch zu modellieren, während dieser bei anderen Teilen – etwa den Fenstern – stationär modelliert wird.

3.1.3 Typischer Aufbau von Norm-Rechenverfahren

Die in den Normen zur Berechnung von Energiekennzahlen konkret verwendeten *physikalischen* Modelle und Verfahren sind in ➪ Abschnitt 2.1.3.1 beschrieben. Hier noch ein paar Worte zu deren *methodischer* Umsetzung:

Allgemein sind die Rechenverfahren in typischen Normen in Form von *eindeutig vorgegebenen Rechenwegen* definiert, die sich in einzelne Rechenschritte gliedern, d.h. die einzelenen Rechenschritte stehen mathematisch gesehen in einem *kausalen* Zusammenhang zueinander, auch wenn die physikalischen Kausalitäten nicht immer so eindeutig vorhanden sind. Die einzelnen Rechenschritte sind üblicherweise als Gleichungen, Tabellen oder Diagramme definiert.

Aus dieser quasi allgemein gültigen Grundstruktur leitet sich die Architektur des entwickelten Frameworks ab, die im folgenden Kapitel ausführlich beschrieben wird.

3.2 Grundelemente der Simulationsumgebung

Ähnlich wie auch die meisten anderen normativ festgelegten Berechnungsverfahren im Bauwesen setzen sich die Verfahren zur Ermittlung von Energiekennzahlen im Wesentlichen aus zwei Typen von Rechenschritten zusammen: Formeln (Gleichungen) und Tabellen. Vereinzelt gibt es auch Diagramme - diese lassen sich aber entweder als analytische Formel oder als Tabelle mit diskreten Werten abbilden, stellen also eigentlich keinen eigenständigen Typ dar.
Weiters sind im Berechnungsablauf an verschiedenen Stellen Fallunterscheidungen nötig, die ebenfalls als eine Art Rechenschritt abgebildet werden müssen (z.B.: Wohngebäude-Nichtwohngebäude, Heizfall-Kühlfall, Fensterlüftung-Raumlufttechnik). Nicht zuletzt ist es erforderlich, dass Schnittstellen für die Übergabe von Werten zwischen den einzelnen Rechenschritten geschaffen werden.

Die Berechnungsmethoden sind mathematisch nicht besonders anspruchsvoll – entsprechend einfach gestaltet sich deren mathematische Abbildung in Modelica. Der weitaus größere Aufwand in der Programmierung von Rechenschritten besteht in der Deklaration von Schnittstellen (Konnektoren) zur Übernahme von Ergebnissen aus voran gehenden Rechenschritten, bzw. zur Übergabe des Ergebnisses an nachfolgende Rechenschritte. Ebenfalls relativ aufwändig ist die Formulierung von Annotationen für jene graphischen Symbole, durch welche das jeweilige Element später im Blockdiagramm repräsentiert werden soll.

Es ist also nahe liegend, Basisklassen (Prototypen) für verschiedene Arten von Elementen zu formulieren, die standardisierte Icons, Schnittstellen, Variablen, etc. enthalten, und von denen sich dann die konkreten Formeln, Tabellen, Fallunterscheidungen, etc. ableiten. Die wesentlichsten dieser Prototypen sind in folgender Tabelle zusammengefasst:

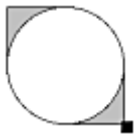
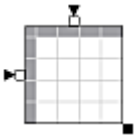
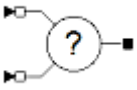
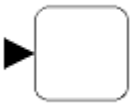
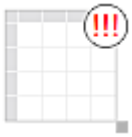
Icon	Kurzbeschreibung der Klasse
	*OIB_RL6.Prototypen.Gleichung* und -*Gleichung_vektoriell* (➪ siehe Abschnitt 3.2.2) Basisklassen für Gleichungen (Formeln) mit skalarem bzw. vektoriellem Ergebnis. Beide Klassen haben das selbe graphische Symbol, aber bei *Gleichung_vektoriell* ist die Schnittstelle für den Ergebniswert ein 1-dimensionals Konnektor-Array (➪ siehe Abschnitt 3.2.2.2).
	*OIB_RL6.Prototypen.Basistabelle*, -*Basistabelle_1D*, -*Basistabelle_2D* und , -*Tabelle* (➪ siehe Abschnitt 3.2.3) Basisklassen für verschiedene Arten von Tabellen. Basistabellen beinhalten ein Grundgerüst für Tabellen mit 0, 1 oder 2 Eingangswerten. *Tabelle* beinhaltet ein Grundgerüst für eine Tabelle mit 2 Eingangswerten sowie Funktionen zur Interpolation, Extrapolation sowie für komplexe Spalten- bzw. Zeilenindexierung.
	*OIB_RL6.Prototypen.Fallunterscheidung* (➪ siehe Abschnitt 3.2.4.2) Basisklasse für Unterscheidung zwischen verschiedenen Rechenwegen (z.B.: Wohngebäude-Nichtwohngebäude, Heizfall-Kühlfall, Fensterlüftung-Raumlufttechnik)
	OIB_RL6.Prototypen.iconSubmodell Basisklasse für Submodelle. *iconSubmodell* beinhaltet keine Schnittstellen, sondern nur das graphische Symbol. Optional kann das Icon mittels einer von *OIB_RL6.Prototypen.iconSubmodellBackground* abgeleiteten Klasse farbig hinterlegt werden.
	*OIB_RL6.Prototypen.inputConnector* (➪ siehe Abschnitt 3.2.4.1) Basisklasse für Schnittstellen für Eingangswerte. Der Konnektor beinhaltet einen skalaren, einheitslosen Real-Wert, der bei Bedarf mit einer physikalischen Einheit redeklariert werden kann. Bei vektoriellen Eingangswerten ist ein Array aus solchen Konnektoren zu erzeugen.
	OIB_RL6.Prototypen.outputConnector (➪ siehe Abschnitt 3.2.4.1) Basisklasse für Schnittstellen für Ausgangswerte - also das Pendant zu *inputConnector*. Auch dieser Konnektor beinhaltet einen skalaren, einheitslosen Real-Wert, der bei Bedarf mit einer physikalischen Einheit redeklariert werden kann. Bei vektoriellen Ergebnissen ist ein Array aus solchen Konnektoren zu erzeugen (➪ siehe Abschnitt 3.2.2.2).
	OIB_RL6.Prototypen.iconOverlayEmbeddedTable Im Fall von eingebetteten Tabellen (die ansonsten graphisch nicht sichtbar wären) soll mittels Mehrfachvererbung von dieser Klasse ein Icon-Overlay vererbt werden, das links unten über das eigentliche Icon der jeweiligen Klasse gelegt wird (➪ siehe Listing 3.2 Zeile 4).
	*OIB_RL6.Prototypen.iconOverlayTodo* Im Fall von Klassen, an denen noch Verbesserungen nötig sind, soll mittels Mehrfachvererbung von dieser Klasse ein Icon-Overlay vererbt werden, das rechts oben über das eigentliche Icon der jeweiligen Klasse gelegt wird (➪ siehe Listing 3.2 Zeile 5).

Tabelle 3.1: Wesentliche Grundelemente des entwickelten Frameworks (Prototypen)

3.2.1 Rechenschritte

Konkret sieht die Klassenhierarchie der Basisklassen für verschiedene Arten von (nicht vektoriellen) Rechenschritten im gegenständlichen Projekt so aus:

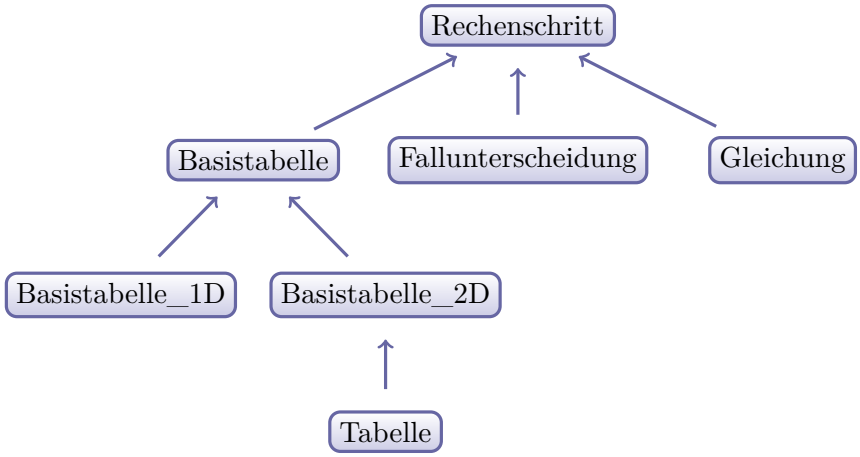


Abbildung 3.5: Klassenhierarchie von Rechenschritten mit skalarem Ergebnis

In der Klasse *Rechenschritt* wird nur eine standardisierte Schnittstelle (der Konnektor *result*) zur Übergabe des Ergebniswertes (*result.value*) an nachfolgende Rechenschritte formuliert. Von *Rechenschritt* leiten sich dann die Prototypen für unterschiedliche Arten von Rechenschritten ab.

Da es (im Widerspruch zur Modelica Language Specification [Mod10], Kapitel 7.3.3) in keiner einzigen der untersuchten Modelica-Implementationen (*OpenModelica*, *MathModelica* und *Dymola*) möglich ist, bei Redeklarationen von Variablen deren Dimension zu ändern, ist es nötig, eine eigene Basisklasse für Gleichungen mit einem vektoriellen Ausgangskonnektor zu erstellen: die Klasse *Rechenschritt_vektoriell*.

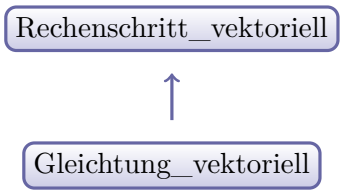


Abbildung 3.6: Klassenhierarchie von Rechenschritten mit vektoriellem Ergebnis

3.2.2 Formeln (Gleichungen)

3.2.2.1 Die Basisklasse *Gleichung*

Diese Klasse ist der Prototyp für Gleichungen mit skalaren Ergebnissen. Ihr vollständiger Bezeichner lautet *OIB_RL6. Prototypen. Gleichung*. Sie leitet sich von *Rechenschritt* ab und erweitert diese Basisklasse um die Annotation zur Formulierung des graphischen Grundsymbols.

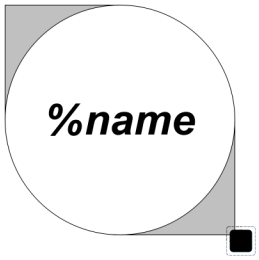


Abbildung 3.7: graphisches Symbol einer Gleichung, gerendert in *MathModelica* 3.0

Abbildung 3.7 zeigt das Grundsymbol einer Gleichung. Rechts unten sieht man die Schnittstelle zur Übergabe des Ausgangswertes (von *MathModelica* strichliert eingerahmt, um Redeklarierbarkeit zu kennzeichnen). Die graue Fläche kann durch eine andersfarbige Fläche ersetzt werden, um bei komplexeren Blockdiagrammen Gleichungen visuell besser gruppieren zu können. Der Text *%name* wird bei Gleichungs-Objekten durch den Namen der jeweiligen Instanz ersetzt.

3.2.2.1.1 Beispiel: ÖNORM B 8110-6 (2007), Gleichung 21 (entspricht ÖNORM B 8110-6 (2010), Gleichung 12)

Die Formel für den pauschalen Leitwertzuschlag für Wärmebrücken im Kühlfall gem. ÖNORM B 8110-6 lautet

$$L_\psi + L_\chi = 0{,}2 \cdot \left(0{,}75 - \frac{\sum_i f_{FH,i} \cdot f_{i,c} \cdot A_i \cdot U_i}{\sum_i A_i} \right) \cdot \sum_i f_{FH,i} \cdot f_{i,c} \cdot A_i \cdot U_i \geq 0{,}1 \cdot L_e \quad (3.1)$$

Die Umsetzung dieser Formel als Klasse in Modelica erfolgt folgendermaßen:

```
1 class '(21) L_psi_plus_L_chi'
2   extends OIB_RL6.Prototypen.Gleichung
3   (redeclare OIB_RL6.Schnittstellen.outputConnectorLeitwert result);
4
5     parameter Real f[:] =
6       OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.f
```

```
7       "Temperaturkorrekturfaktoren [-]";
8     parameter Real f_FH[:] =
9       OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.f_FH
10      "Temperaturkorrekturfaktoren für Flächenheizungen [-]";
11    parameter Modelica.SIunits.Area A[:] =
12      OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.A
13      "Bauteilflächen [m^2]";
14    parameter Modelica.SIunits.CoefficientOfHeatTransfer U[:] =
15      OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.U
16      "Wärmedurchgangskoeffizienten [W/(m^2*K)]";
17
18    Modelica.SIunits.ThermalConductance L_T;
19
20  equation
21    L_T = sum( f_FH[j] .* f[j] .* A[j] .* U[j]);
22    result.value = max(0.1, 0.2 * (0.75 - L_T / sum(A)) * L_T);
23
24 end '(21) L_psi_plus_L_chi';
```

Listing 3.1: ÖNORM B 8110-6 (2007), Formel 21 als Klasse in Modelica

Die Zeilen 1 und 24 deklarieren bzw. begrenzen die Klasse. Dazu ist anzumerken, dass die Anwendung der Q-IDENT Notation (sog. Quoted Identifier - also Bezeichner, die Sonder- und/oder Leerzeichen enthalten, und deshalb zwischen Hochkommas gestellt werden (➪ [Mod07], Kap. 2.3.1) wegen Bugs in einigen Modelica-Umgebungen unter Umständen zu Problemen führen kann. (➪ siehe Abschnitt A.3)
In den Zeilen 2 und 3 wird *OIB_RL6.Prototypen.Gleichung* als Basisklasse definiert, und deren *result*-Konnektor wird so redeklariert, dass seine physikalische Einheit jener des Ergebnisses der Gleichung entspricht.
In den Zeilen 5 bis 16 werden jene Größen deklariert, die als gebäudeabhängige Parameter in Gleichung 21 verwendet werden, und es werden ihnen Defaultwerte aus *OIB_RL6.DefaultProjekt* zugewiesen.
Die eigentliche Gleichung ist in den Zeilen 20 bis 22 formuliert, wobei jener Teil, welcher mehrfach in Gleichung 21 vorkommt, als separat ermittelter Hilfswert in die Gleichung eingeht. Dieser Hilfswert wird zuvor in Zeile 18 deklariert.

3.2.2.2 Die Basisklasse *Gleichung_vektoriell*

Diese Basisklasse entspricht *Gleichung*, leitet sich aber von *Rechenschritt_vektoriell* ab, und hat somit ein 1-dimensionales Array aus *result*-Konnektoren als Ausgangsschnitt-

stelle. Ihr vollständiger Bezeichner lautet *OIB_RL6. Prototypen. Gleichung_vektoriell*. Dem Autor ist zwar bis jetzt noch nie eine „echt“ vektorielle Gleichung in einer bautechnischen Norm aufgefallen, aber vektorisierte Formulierungen sind z.B. überall dort naheliegend, wo in einer Gleichung über eine größere bzw. über eine nicht näher spezifizierte Anzahl von Elementen summiert wird.

3.2.2.2.1 Beispiel: ÖNORM B 8110-6 (2007), Gleichung 56b (entspricht ÖNORM B 8110-6 (2010), Gleichung 47b)

Die Formel für solare Wärmegewinne von Nicht-Wohngebäuden im Kühlfall gem. ÖNORM B 8110-6 lautet

$$Q_{s,h} = \sum_j \left(I_{S,j} \cdot \sum_k A_{trans,c,k,j} \right) + \sum_j \left(\sum_k Q_{opak,c,k,j} \right) \tag{3.2}$$

Diese Gleichung wird folgendermaßen umformuliert[1]:

$$Q_{s,h} = \sum_k \left(I_{S,k} \cdot A_{trans,c,k} \right) + \sum_k \left(Q_{opak,c,k} \right) \tag{3.3}$$

Die Umsetzung dieser Gleichung als Klasse in Modelica erfolgt so[2]:

```
1 class '(56b) Q_sc'
2   extends OIB_RL6.Prototypen.Gleichung
3    (redeclare OIB_RL6.Schnittstellen.outputConnectorWaermemenge
        result);
```

1 Die Formulierung von Gleichung 56b (und einigen anderen) in ÖNORM B 8110-6 (2007) mit verschachtelten Summen liegt wahrscheinlich darin begründet, dass die Norm ein diskretes Raster für Orientierungen (Azimut und Höhenwinkel) vorgibt – dadurch wird die Anzahl an möglichen Ausrichtungen von Flächen endlich, und es kann über die Ausrichtungen summiert werden.
Weil die Gesamtenergie der Globalstrahlung I_S auf eine Flächeneinheit aber eigentlich von deren Orientierung – also von Orientierung des jeweiligen Fensters – abhängig ist, ist I_S ein Paramter des Fensters, der in die innere Summe „hineingezogen“ werden soll. Im Term für die opaken Bauteile macht die Verschachtelung der Summen überhaupt keinen Sinn, weil der äußeren Summe kein einziger Parameter zugeordnet ist. Die Formel ist also nicht nur unnötig kompliziert, sondern sie erlaubt auch nur Simulationsmodelle, bei denen die Orientierung an ein diskretes Raster gebunden ist. Vor dem Hintergrund der Aufgabenstellung im gegenständlichen Projekt ist sie also ungeeignet.

2 Im hier gedruckten Listing sind zur besseren Lesbarkeit die meisten Annotationen und Kommentare weggelassen. Außerdem wurde Zeile 5 zur Demonstration von Erläuterungen in ➪ Tabelle 3.1 eingefügt.

```
4     extends OIB_RL6.Prototypen.iconOverlayEmbeddedTable();
5     extends OIB_RL6.Prototypen.iconOverlayTodo(); // hier: nur als
          Demo
6
7     constant Integer dimension_transparent =
8       size(OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.A_g,1);
9     constant Integer dimension_opak =
10      size(OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.A_opak,1);
11
12    OIB_RL6.Schnittstellen.inputConnectorStrahlungsumme I_S;
13    OIB_RL6.Schnittstellen.inputConnectorFlaeche
          A_transc[dimension_transparent];
14    OIB_RL6.Schnittstellen.inputConnectorWaermemenge
          Q_opak[dimension_opak];
15
16    OIB_RL6.OENORM_B8110_5.Tabellen.Tabelle_E Tabelle_E;
17    OIB_RL6.OENORM_B8110_6.Gleichungen.TerminatorInteger Terminator;
18
19    parameter Real[dimension_transparent] azimut =
20      OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.azimut;
21    parameter Real[dimension_transparent] neigung =
22      OIB_RL6.DefaultProjekt.gebaeude.bauteile.gegenAussen.neigung;
23
24    Integer altitudeIndex =
25      OIB_RL6.OENORM_B8110_5.Tabellen.labels.tabE_altitudeIndex
26        (OIB_RL6.DefaultProjekt.allgemein.standort.H);
27    Integer neigungIndex[dimension_transparent] =
28      OIB_RL6.OENORM_B8110_5.Tabellen.labels.tabE_neigungIndex(neigung);
29    Integer azimutIndex[dimension_transparent] =
30      OIB_RL6.OENORM_B8110_5.Tabellen.labels.tabE_azimutIndex(azimut);
31
32    Real transpositionsFaktor[dimension_transparent];
33
34  equation
35
36    connect(Terminator.result,Tabelle_E.in_x);
37    connect(Terminator.result,Tabelle_E.in_y);
38    connect(Terminator.result,Tabelle_E.neigung);
39    connect(Terminator.result,Tabelle_E.azimut);
40
41    for i in 1:dimension_transparent loop
42      transpositionsFaktor[i] = Tabelle_E.data
43       [altitudeIndex,
44        OIB_RL6.Funktionen.timeConversion.currentMonth(time),
```

```
45          neigungIndex[i],
46          azimutIndex[i]];
47      end for;
48
49      result.value
50        = sum((I_S.value * transpositionsFaktor) .* A_transc.value)
51        + sum(Q_opak.value);
52
53  end '(56b) Q_sc';
```

Listing 3.2: ÖNORM B 8110-6 (2007), Formel 56b als Klasse in Modelica

Die Zeilen 12 bis 40 sind an dieser Stelle (noch) nicht interessant. Speziell was die Implementationen von Tabelle E und der Schnittstellen betrifft, sei hier auf ➪ Abschnitt 3.2.3 verwiesen. Im Zusammenhang mit der oben erwähnten Summenbildung mittels vektorieller Formulierung sind die Zeilen 7 bis 10 sowie 41 bis 51 interessant: In den Zeilen 7 bis 10 werden die Dimensionen jener Vektoren, über deren Elemente später summiert wird, anhand der Gebäudeparameter (vektorielle Listen der Fenster und der opaken Bauteile) festgelegt – diese Dimension ist die Anzahl der Elemente, über die dann in den beiden Termen von Gleichung 3.3 summiert wird.
In den Zeilen 41 bis 47 wird in einer for-Schleife der Vektor *transpositionsFaktor* aus Werten der Tabelle E erzeugt. In Zeile 50 wird dieser Vektor dann mit dem (Skalar-)Wert der mittleren Monatssummen der Globalstrahlung *I_S.value* multipliziert. Der so erhaltene Vektor der $I_{S,k}$-Werte aus Gleichung 3.3 wird schließlich elementweise mit dem Vektor der solar wirksamen Kollektorflächen *A_transc.value* multipliziert, und man erhält den Vektor der einzelnen Elemente, über die schließlich mittels der `sum()`-Funktion summiert wird. In Zeile 50 wird also der Term $\sum (I_S \cdot A_{trans,c})$ aus Gleichung 3.3 erzeugt. Der Term $\sum (Q_{opak,c})$ wird analog dazu in Zeile 51 erzeugt.
In ➪ Abbildung 3.9 wird übrigens das gerenderte Symbol der Gleichung mit ihren in Zeile 12 bis 14 deklarierten Eingangs-Schnittstellen *I_S*, *A_transc* und *Q_opak* sowie den in den Zeilen 4 und 5 verwendeten Icon-Overlays dargestellt.

3.2.3 Tabellen

3.2.3.1 Grundsätzliches

Tabellen werden ebenfalls als Klassen implementiert, die im Wesentlichen folgende Elemente enthalten:

- Schnittstelle für den Tabellen-Rückgabewert in Form des Output-Konnektors *out*
- Schnittstellen für den Spalten- und Zeilen-Eingangswert in Form der Input-Konnektoren *in_x* und *in_y*
- Kopfzeile bzw. Indexspalte in Form der Vektoren *header_x*[1] und *header_y*[1] (bzw. optional in Form der Enumerationen *label_x*[2]. und *label_y*[2]).
- Tabellenwerte in Form des Arrays *tableData* (ein- oder zweidimensional)
- Routinen zur Ermittlung des Spalten- bzw. Zeilenindex aus den Eingangswerten[1]
- Routinen zur Inter- bzw. Extrapolation[1]

Der Zugriff auf die Tabellendaten über Konnektoren der Tabellen-Klasse entspricht dem für objektorientierte Programmierung typischen Prinzip der Kapselung von Daten und macht vor allem Sinn, wenn

1. die Tabellen-Eingangswerte *nicht* unmittelbar als Indizes zum Adressieren einzelner Werte innerhalb des Arrays der Tabellendaten herangezogen werden können, und/oder wenn
2. die Tabellen-Ausgangswerte *nicht* unmittelbar aus einzelnen Werten innerhalb des Arrays der Tabellendaten ermittelt werden.

Ist keine dieser beiden Voraussetzungen gegeben (d.h. die Array-Indizes stehen unmittelbar zur Verfügung und die Array-Werte sind die unmittelbaren Ausgangswerte einer Tabellenabfrage), dann macht es unter Umständen Sinn, die Abfrage unter Umgehung der Schnittstellen direkt auf die Daten des *tableData*-Arrays zu „setzen“.

Dazu keine Instanz der Tabelle im Modell zu erzeugen, sondern direkt auf die Daten der Klasse zuzugreifen, wäre zwar theoretisch in *Dymola* möglich[3], ist aber nicht konform zu dem Modelica-Sprachstandard [Mod07], Kap. 5.3.2, und wird deshalb nicht gemacht.

Stattdessen kann eine Instanz der Tabelle innerhalb der Gleichung erzeugt werden, in welcher die Tabellendaten benötigt werden (➪ siehe Listing 3.2 Zeile 16ff). Auf die

1 Nur bei Tabellen, die auf der Basisklasse *Tabelle* aufbauen (➪ siehe Abschnitt 3.2.3.3), oder die unabhängig davon entsprechend erweitert wurden.
2 Nur bei Tabellen, die auf der Basisklasse *BasisTabelle_1D* oder *BasisTabelle_2D* aufbauen (➪ siehe Abschnitt 3.2.3.2)
3 In *Dymola* kann auf Daten innerhalb einer Nicht-`package`-Klasse zugegriffen werden, indem die Umgebungsvariable `Advanced.CheckPackageRestriction:=false;` gesetzt wird.

Daten dieser Instanz kann dann entweder (mittels Konnektoren) über die „normalen“ Schnittstellen der Tabelle zugegriffen werden – z.B., wenn die Lookup-Methoden der Klasse verwendet werden sollten –, oder es kann direkt aufs *tableData*-Array des Objektes zugegriffen werden. (In dem Fall müssen die Indizes zum Adressieren der benötigten *tableData*-Werte natürlich vorher explizit ermittelt werden.)

☞ **Hinweis:** Vor dem Hintergrund des eigentlichen Zieles der Entwicklung – einer Kopplung von vereinfachten an simulative Verfahren – muss man sich also bei der Entscheidung, eine Tabelle direkt in eine Gleichung einzubetten, folgender Einschränkung bewusst sein: Durch die Tatsache, dass die Tabelle dann aber nicht mehr als „sichtbares“ Objekt im Modell repräsentiert ist, ist es auch nicht möglich, die Tabelle durch jenes physikalische Modell zu ersetzen, welches der Tabelle evtl. zugrunde liegt.

3.2.3.2 Die Basisklassen *BasisTabelle_1D* und *BasisTabelle_2D*

Diese Klassen sind Prototypen von trivialen 1D- bzw. 2D-Tabellen. Ihre vollständigen Bezeichner lauten *OIB_RL6. Prototypen. BasisTabelle_1D* bzw. *OIB_RL6. Prototypen. BasisTabelle_2D*. Unter trivialen Tabellen verstehen sich Tabellen, deren Kopfzeile bzw. Indexspalte keine Werte*bereiche* enthalten und bei denen weder interpoliert noch extrapoliert werden muss.

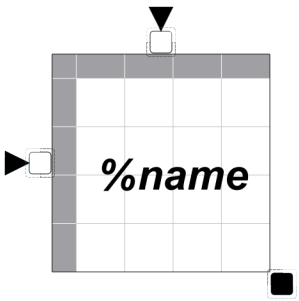


Abbildung 3.8: graphisches Symbol einer 2D-Tabelle, gerendert in *MathModelica* 3.0

Wie das Ergebnis einer Tabellenabfrage ermittelt wird, ist bei der Verwendung dieser Basisklassen dem Programmierer überlassen. Sehr kleine Tabellen – wie z.B. die Tabelle 13 in ÖNorm B8110-6:2007, die nur 2 Werte enthält – können z.B. unter Verwendung von if-then-else Konstrukten implementiert werden, ohne dass ein Array verwendet wird.

Wenn die Tabellendaten als Array implementiert sind, die Array-Indizes unmittelbar zur Verfügung stehen, und die Array-Werte unmittelbar als Ausgangswerte von Tabellenabfragen herangezogen werden, dann kann die Tabelle auch ohne Instanziierung eines Tabellen-Objektes verwendet werden (➪ siehe Abschnitt 3.2.3.1).

Die Klassen beinhalten das Grundgerüst für 1- bzw. 2-dimensionale Tabellen. 1D-Tabellen enthalten nur Tabellenwerte in Form *einer* Zeile bzw. *einer* Spalte sowie die

zugehörige Kopfzeile bzw. Indexspalte. 2D-Tabellen enthalten eine Kopfzeile, eine Indexspalte sowie ein zweidimensionales Array aus Tabellenwerten. Das Grundgerüst besteht im Wesentlichen aus folgenden Elementen:

- *out*: Der Output-Konnektor dient zur Übergabe des Rückgabewertes (Typ *Real*).
- *in_x* und *in_y*[1]: Diese Input-Konnektoren dienen jeweils zur Übernahme eines Tabellen-Eingangswertes vom Typ *Real* zur Indexierung der Spalte bzw. Zeile.

  ☞ **Hinweis:** Werte vom Typ *Real* können nicht direkt zur Indexierung von Arrays herangezogen werden. Wenn ein Array mit den Eingangswerten indexiert werden soll, so muss entweder der Konnektor als *Integer*-Konnektor redeklariert werden, oder der Eingangswert muss vor der Verwendung auf Typ *Integer* gecastet werden.
- *label_x* und *label_y*[1]: Diese Enumerationen enthalten sprechende Bezeichner für die Kopfzeile bzw. die Indexspalte. Mittels der Enumeration und der Integer-Funktion kann man auf einfache Weise zum Index einer Spalte bzw. Zeile gelangen. Neben der Möglichkeit, Indizes auf diese Art mit sprechenden Bezeichnern und ausführlichen Kommentaren zu versehen, besteht ein weiterer Vorteil der Verwendung von Enumerationen zur Index-Generierung darin, dass bei einer Umordnung der Tabelle (so wie sie z.B. bei Tabelle 2 aus ÖNORM B8110-5 in der Fassung von 2010 stattgefunden hat) keine Änderungen außerhalb der Tablellenklasse vorgenommen werden müssen. Die direkte Verwendung von Enumerationen zur Array-Indexierung ist in Modelica leider dzt. praktisch noch nicht möglich (➪ siehe Abschnitt A.3).

  Die beiden Typen haben keine „vitale“ Funktion und können – z.B., wenn das Prinzip auf eine bestimmte Tabelle nicht sinnvoll anwendbar ist – ignoriert werden.
- *tableData*: Dieses Array ist der Container der eigentlichen Tabellenwerte. Es kann ignoriert werden, wenn es zur Ermittlung der Tabellendaten nicht benötigt wird (z.B. mittels if-then-else-Konstrukt).

1 nur bei 2D-Tabellen

☞ **Hinweis:** 2D-Arrays werden in Modelica als Serie von Spaltenvektoren eingegeben. Der Quelltext ist aber besser lesbar, wenn die Tabellendaten zeilenweise eingegeben werden. In dem Fall ist entweder das eingegebene Array vor der Verwendung mit der Funktion *transpose()* zu transponieren, oder es sind beim Lookup der x- und y-Index zu vertauschen

- Annotationen zur graphischen Darstellung (➪ siehe Abbildung 3.8)

3.2.3.3 Die Basisklasse *Tabelle*

Die Basisklasse *Tabelle* ist der Prototyp einer nicht-trivialen 2D-Tabelle. Ihr vollständiger Bezeichner lautet *OIB_RL6. Prototypen. Tabelle*. Neben dem Grundgerüst einer trivialen 2D-Tabelle vom Typ *Basistabelle_2D* (➪ siehe Abschnitt 3.2.3.2) stellt die Klasse *Tabelle* zusätzlich verschiedene Methoden zur Spalten- und Zeilenindexierung sowie zur Interpolation und Extrapolation zur Verfügung:

- *extrapolate*: Der Parameter ist nur wirksam, wenn interpoliert wird. Es ist ein Array aus vier Boolean-Werten, das mittels des Enumerationstyps *OIB_RL6. Datentypen. tableBorder* indexiert ist. Mittels der Werte in *extrapolate* wird festgelegt, ob – und wenn ja: über welche Tabellenränder hinaus – extrapoliert werden soll.
- *method_x* und *method_y*: Diese Parameter legen die Art der Spalten- bzw. Zeilenindexierung fest. Er ist ein Enumerationstyp, der in *OIB_RL6. Datentypen. table_lookup_method* deklariert ist.

  method_x und *method_y* können folgende Werte annehmen:

  table_lookup_method.nearest: Es wird auf jenen Indexwert gesprungen, bei dem der Betrag der Differenz zum Eingangswert möglichst gering ist („best match“). Bei Eingangswerten, die exakt zwischen zwei Indexwerten liegen, wird auf den Indexwert mit dem niedrigeren Index gesprungen.

  Beispiel:

  Beispiel:

  Tabelle:

15	20	25
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
17.5	0
9	0
27	80

  table_lookup_method.index: Der Eingangswert wird auf ein Integer gecastet und direkt als Index verwendet. Die Methode ist sehr effizient.

Beispiel:

Tabelle:

15	20	25
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
2	40
2.9	40
4	Error

table_lookup_method.exact: Es wird auf jenen Indexwert gesprungen, dessen Wert genau dem Eingangswert entspricht.

Beispiel:

Tabelle:

15	20	25
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
15.1	Assertation
15.0	0
25	80

table_lookup_method.left: Diese Indexierungsmethode erlaubt es, Wertebereiche in der Kopfzeile bzw. der Indexspalte zu definieren, indem *header_x* bzw. *header_y* ein Element mehr beinhalten, als die Tabelle Spalten bzw. Zeilen hat. Der „linke“ Nachbarwert wird noch zum Wertebereich gezählt.

Beispiel:

Tabelle:

1-10	10-100	100-1000
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
10	40
50	40
1000	80

table_lookup_method.right: Diese Indexierungsmethode funktioniert im Prinzip wie *table_lookup_method.left*, aber der „rechte“ Nachbarwert wird noch zum Wertebereich gezählt.

Beispiel:

Tabelle:

1-10	10-100	100-1000
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
10	0
50	40
1000	80

table_lookup_method.interpolate: Mit dieser Indexierungsmethode kann entweder spalten- oder zeilenweise interpoliert werden.

Es wird zuerst jener Wert in der Kopfzeile bzw. Indexspalte ermittelt, der dem jeweiligen Eingangswert am nächsten liegt. Dieser Wert ist der erste Stützwert. Als zweiter Stützwert wird der jeweils zweite (also der „weiter entfernte“) Nachbarwert in der Kopfzeile bzw. Indexspalte ermittelt und dann wird zwischen diesen beiden Werten linear interpoliert.
Wenn der erste Stützwert am Rand der Kopfzeile bzw. Indexspalte liegt, dann wird aus den beiden dem Eingangswert am nächsten liegenden Stützwerten extrapoliert, sofern Extrapolation am jeweiligen Tabellenrand zulässig ist. (Dies wird mittels Setzens des entsprechenden *extrapolate*-Wertes auf `true` festgelegt.) Andernfalls erfolgt ein Assertion-Error-Call.
Die Interpolation und Extrapolation erfolgen immer linear. Es kann also nur zwischen Spalten *oder* zwischen Zeilen interpoliert werden (nicht zwischen Spalten *und* Zeilen), d.h., eine gleichzeitige Zuweisung von *table_lookup_method.interpolate* an *method_x* *und* *method_y* ist nicht möglich.

Beispiel:

Tabelle:

15	20	25
:	:	:
0	40	80
:	:	:

Spalteneingangswert:	Tabellenausgangswert:
17.5	20
0	-40
35	Assertation

In o. a. Beispiel ist der *extrapolate*-Werte für den linken Tabellenrand auf `true`, und für den rechten Tabellenrand auf `false` gesetzt.

☞ **Hinweis:** Die Performanz von Tabellen, die auf der Basisklasse *Tabelle* aufbauen, ist derzeit schlecht, weil bei der Programmierung mehrere Bugs in *OpenModelica* und *MathModelica*, die sich negativ auf die Geschwindigkeit der Tabellenabfragen auswirken, mit Workarounds umgangen werden mussten:

- In der derzeitigen Programmierung werden alle in der *lookup*-Funktion benötigen Werte explizit als Paramter an die Funktion übergeben. Eigentlich sollten konstante Werte (z.B. *header_x*, *header_y*, *tableData*, *method_x*, *method_y*,

extrapolate) auch ohne explizite Übergabe im Scope der Funktion zur Verfügung stehen. Zur Zeit der Programmierung hat dies aber mit redeklarierten Konstanten nicht funktioniert.

- Dass das *tableData*-Array erst bei der Übergabe an die *lookup*-Funktion transponiert wird, sollte ebenfalls nicht sein, weil so die Transponierung bei jeder Tabellenabfrage stattfindet (anstatt nur einmal bei der Deklaration von *tableData*) Zur Zeit der Programmierung war es aber nicht möglich, das Array bereits bei seiner Deklaration zu transponieren, weil ein Function-Call in der Deklaration einer Variable (z.B. `constant Real x[2,2] = transpose({{1,2},{3,4})`) nicht möglich war.

  Eine Alternative zur Transponierung der Matrix wäre ein Vertauschen der Indizies beim Adressieren des Arrays. Dazu müsste aber an mehreren Stellen in die *lookup*-Funktion eingegriffen werden, und nach Meinung des Autors wäre dadurch der Grundstein für zukünftige Missverständnisse & Bugs gelegt.

- Das *extrapolate*-Array wird dzt. in Form von 4 separaten Boolean-Werten an die *lookup*-Funktion übergeben, weil die Übergabe als Array vom Typ boolean[4] durch einen Bug in *OpenModelica* und *MathModelica* verhindert wird.

Insbes. die beiden ersten o. a. Punkte führen dazu, dass bei jeder Tabellenabfrage unnötig viele Daten transformiert und verschoben werden, was sich im Fall von großen Tabellen sehr negativ auswirkt.

☞ **Hinweis:** Die Programmierung der Klasse *Tabelle* hat zur Zeit von *OpenModelica* 1.5.0 RC1 bzw. *MathModelica* 2.1 stattgefunden. Einige der oben genannten Probleme wurden seither behoben, andere sind dazu gekommen. So ist die Klasse ab Version 1.5.0 RC2 wegen Bugs im Bereich der Array-Indexierung nicht mehr mit *OpenModelica* kompatibel (auch nicht mit 1.5.0 final). In *MathModelica* 3.0 beta und in *Dymola* 7.4 funktioniert die Klasse. Da im Laufe des Projektes auf *Dymola* als Simulationsumgebung umgestellt wurde, sind einige der oben erwähnten „bremsenden“ Workarounds nicht mehr nötig, wurden aber noch nicht geändert. Im Sourcecode finden sich entsprechende Kommentare, die eine spätere Änderung der Klasse erleichtern sollten.

3.2.3.3.1 Beispiel: ÖNORM B 8110-6 (2007), Tabelle 16 (entspricht ÖNORM B 8110-6 (2010), Tabelle 11)

Die Tabelle der Standardwerte für Raumbelastungsgrade bei Abluftleuchten gem. ÖNORM B 8110-6 lautet

Anschlussleistung, in $m^3/h \cdot W$	**0.20**	**0.30**	**0.50**	**1.00**
η_L bei Absaugung über Deckenhohlraum	0.80	0.70	0.55	0.45
η_L bei Absaugung durch gedämmte Leitungen	0.45	0.40	0.35	0.30
η_L bei Absaugung durch ungedämmte Leitungen	0.40	0.35	0.30	0.25

Tabelle 3.2: Tabelle 16 aus ÖNORM B 8110-6 (2007)

Die Umsetzung dieser Tabelle als Block in Modelica erfolgt folgendermaßen:

```
1  class Tab_16
2    extends OIB_RL6.Prototypen.Tabelle(
3        method_x = OIB_RL6.Datentypen.table_lookup_method.nearest
4       ,method_y = OIB_RL6.Datentypen.table_lookup_method.index
5
6       ,redeclare type label_y =
7          OIB_RL6.OENORM_B8110_6.Tabellen.labels.tab16_y
8
9       ,redeclare constant Real header_x[4] = {0.2, 0.3, 0.5, 1.0}
10      ,redeclare constant Real header_y[4] = 1:4
11
12      ,redeclare constant Real tableData[4,4] = transpose({
13       { 0.80,        0.70,         0.55,       0.45 },
14       { 0.45,        0.40,         0.35,       0.30 },
15       { 0.40,        0.35,         0.30,       0.25 },
16       { 1.00,        1.00,         1.00,       1.00 }})
17   );
18 end Tab_16;
```

Listing 3.3: ÖNORM B 8110-6, Formel 21 als Block in Modelica

In den Zeilen 3 und 4 werden die Lookup-Methoden für Zeilen und Spalten zugewiesen: Beim Spalten-eingangswert soll auf die „am ehesten passende“ Spalte gesprungen werden, weil die Norm in Tabelle 16 keine Interpolation vorsieht. Als Zeileneingangswert wird der Zeilenindex verwendet.

In den Zeilen 6 und 7 wird die Enumeration zur Zeilenbeschriftung redeklariert. Aus ihr kann später jeder Zeilenindex generiert werden. Die Deklaration des Enumerations-Typs muss außerhalb der Klasse im *labels*-Paket erfolgen, damit der Typ global zur Verfügung steht.

In den Zeilen 9 und 10 werden die header redeklariert. Dass *header_y* nicht mit dem Enumerations-Typ *label_y* indexiert wird (was sicher die schönste Lösung wäre), liegt daran, dass dieses Feature in den untersuchten Modelica-Umgebungen fehlerhaft implementiert ist (➪ siehe Abschnitt A.3).

In den Zeilen 14 bis 17 wird schließlich das *tableData*-Array, also die eigentlichen Tabellendaten, redeklariert.

3.2.4 Sonstiges

3.2.4.1 Schnittstellen (Konnektoren)

Konnektoren sind prinzipiell kausal formuliert, und die übergebenen *skalaren* Werte sind mit physikalischen Einheiten besetzt.

Aufgrund der vorgegebenen kausalen Zusammenhänge zwischen Rechenschritten in Norm-Berechnungsmethoden (➪ siehe Abschnitt 3.1.3), wird zwischen Input-Konnektoren (weiß mit Pfeilspitze) und Output-Konnektoren (schwarz) unterschieden (➪ siehe Tabelle 3.1).

Die vollständigen Bezeichner der Prototypen lauten *OIB_RL6.Prototypen.inputConnector* und *OIB_RL6.Prototypen.outputConnector*. Die Konnektor-Klassen für verschiedene physikalische Einheiten sind im Paket *OIB_RL6.Schnittstellen* gesammelt.

Für die Übergabe von vektoriellen Größen werden Arrays vom Typ des jeweiligen Konnektors erzeugt (➪ siehe Abschnitt 3.2.2.2).

3.2.4.2 Fallunterscheidungen

Bei Fallunterscheidungen wird abhängig von einem Entscheidungskriterium einer von mehreren möglichen Rechenwegen eingeschlagen. Die Fallunterscheidungen in der implementierten ÖNORM B 8810-6 sind z.B. Wohngebäude-Nichtwohngebäude, Heizfall-Kühlfall und Fensterlüftung-Raumlufttechnik. Der vollständige Bezeichner des Protoypen für Fallunterscheidungen lautet *OIB_RL6.Prototypen.Fallunterscheidung*, und beinhaltet zwei Input- und einen Output-Konnektor sowie die Annotation für das graphische Symbol (➪ siehe Tabelle 3.1).

3.2.4.3 Graphische Symbole: Icons und Icon-Overlays

Beim Design von graphischen Symbolen ist darauf zu achten, dass die Rendering-Fähigkeiten von *Dymola* relativ beschränkt sind und dass es zu Problemen mit Graphik-Annotationen kommen kann, die z.B. in *MathModelica* generiert wurden (➪ siehe Abschnitt A.3).

Im entwickelten Framework wurden die Icons von Gleichungen, Tabellen und Submodellen so gestaltet, dass es jeweils eine Fläche gibt, deren Farbe verändert werden kann, ohne dass dadurch Schriften unlesbar werden können, oder das Icon unkenntlich werden kann. In der Default-Version der Icons ist diese Fläche grau. Mittels Zuweisung anderer Farben an diese Fläche lässt sich die Zugehörigkeit von Gleichungen, Tabellen und Submodellen zu einer bestimmten Gruppe von Elementen visuell abbilden.[1]

Eine Zuweisung von Farben an Ausgangs-Konnektoren – z.B. um deren physikalische Einheit oder die Zugehörigkeit zu einer bestimmten Gruppe von Elementen visuell abzubilden – ist ebenfalls möglich. Ein praktischer Nebeneffekt bei der Verwendung von Farben für Konnektoren ist, dass in *MathModelica* beim Erzeugen einer `connect()`-Gleichung die Farbe der Verbindungslinie automatisch aus der Farbe des Ausgangs-Konnektors generiert wird. In *Dymola* funktioniert dieses Feature leider nur unter bestimmten Umständen.

Eine Übersicht mit graphischer Darstellung der wesentlichen Elemente des entwickelten Frameworks findet sich in ➪ Tabelle 3.1.

Besonders zu erwähnen sind die Icon-Overlays *iconOverlayEmbeddedTable* und *iconOverlayTodo*: Sie können mittels Mehrfachvererbung in beliebige Klassen abgeleitet werden (➪ siehe Listing 3.2, Zeilen 4 und 5). Dadurch werden die entsprechenden Symbole über das bestehende Icon der jeweiligen Klasse gelegt, um auf ansonsten „unsichtbare“ Besonderheiten in der Programmierung hinzuweisen (➪ siehe Abbildung 3.9).

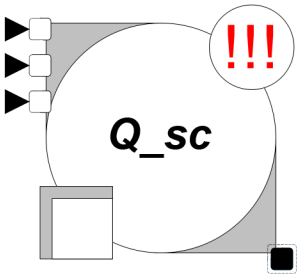


Abbildung 3.9: Icon einer Gleichung mit Icon-Overlays, gerendert in *MathModelica* 3.0

1 Der Autor hat nur bei Submodellen von dieser Möglichkeit Gebrauch gemacht. Andere Projektbeteiligte nutzen die Option auch für Gleichungen und Tabellen.

3.2.4.4 Default-Werte für Parameter

Parameter sind jene Eingangswerte ins Simulationsmodell, die das konkrete Projekt (Gebäude, Gebäudetechnik, Standort, etc.) definieren. In einer Klasse, die Parameter-Werte verwendet, soll diesen schon bei der Deklaration ein Default-Wert zugewiesen werden, um das „Default-Gebäude“ ohne vorherige Initialisierung der Parameter simulieren zu können.

Das Set dieser Default-Parameter ist im Paket *OIB_RL6.DefaultProjekt* festgelegt.

3.2.4.5 Globale Funktionen

Neben den lokalen Methoden in Klassen (Gleichungen/Tabellen) gibt es eine Reihe von global benätigten Funktionen, die von verschiedensten Stellen aus aufgerufen werden.

Hier sind insbes. die Funktionen zum Zeit-Handling zu erwähnen, die benötigt werden, um das 8760-Stunden-Zeitraster für einen Jahr in ein 12-Monats-Zeitraster umzurechnen. Diese sind in *OIB_RL6.Funktionen.timeConversion* zusammengefasst.

4 Anwendung

Im Rahmen dieser Arbeit wurden die ÖNORMEN B 8110-5 und B 8110-6 (➪ siehe Abschnitt 2.1.3.1) auf Basis des entwickelten Frameworks implementiert. Weiters wurde ein dynamisches Modell eines unbeheizten Pufferraumes entwickelt, dessen Temperatur sich abhängig von den angrenzenden Gebäudezonen und der Außenluft einstellt. Neben dem Standard-Klimamodell nach ÖNORM B 8110-5 (2007) wurden auch das in Anhang C der ÖNORM B 8110-5 (2010) neu eingeführte halbsynthetische Klimamodell sowie eine Schnittstelle zur Verwendung von Klimadaten aus Testreferenzjahren entwickelt.

4.1 Die ÖNORM B 8110-5

Die ÖNORM B 8110-5 beinhaltet Klimamodelle für die unterschiedlichen Klimazonen des österreichischen Bundesgebietes und Nutzungsprofile für verschiedene Gebäudetypen. Die Norm besteht hauptsächlich aus Tabellen und enthält nur wenige einzelne Formeln, die keinen Rechenweg im eigentlichen Sinn ergeben, d.h., aus der ÖNORM B 8110-5 ergibt sich (im Gegensatz zu den anderen Normen zur Energiekennzahl-Berechnung) kein Rechenablauf, der in Form von miteinander verknüpften Rechenschritten darstellbar ist.

4.2 Die ÖNORM B 8110-6

In der ÖNORM B 8110-6 ist der Rechengang zur Ermittlung von Heizwärme- und Kühlbedarf festgelegt. Der Rechenweg gliedert sich mehrere, klar voneinander abgrenzbare Teilschritte, die als Teilmodelle unter Verwendung der erstellten Simulationsumgebung umgesetzt wurden. Im Folgenden werden die wesentlichen Teilmodelle der ÖNORM B 8110-6 kurz dargestellt.

4.2.1 Wärmeverluste Q_l

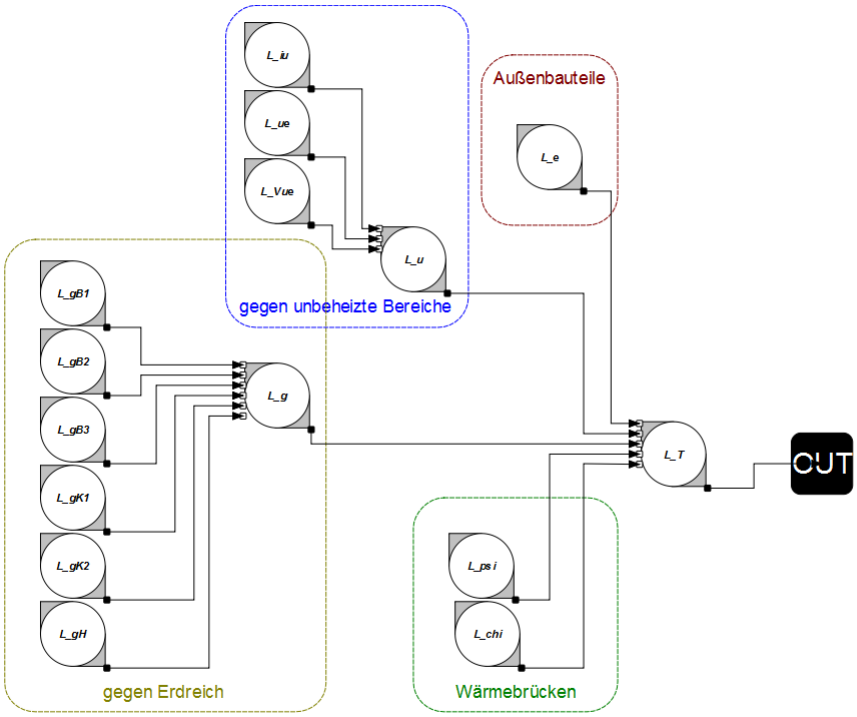
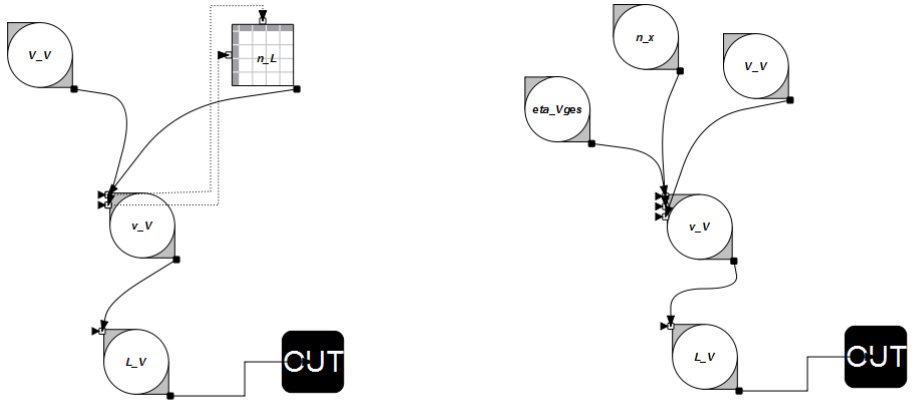


Abbildung 4.1: Modell des Transmissionsleitwertes L_T

Der Transmissionsleitwert L_T setzt sich aus den Leitwerten für Bauteile gegen Außenluft L_e, gegen unbeheizte Gebäudeteile L_u und von erdberührten Bauteilen L_g zusammen. Dazu kommen Leitwerte für punktförmige und lineare Wärmebücken L_ψ und L_χ, die in Abhängigkeit der Leitwerte der Bauteile pauschal formuliert sind.



(a) mit Fensterlüftung $L_{V,FL}$

(b) mit raumlufttechnischen Anlagen $L_{V,RLT}$

Abbildung 4.2: Modelle der Lüftungsleitwerte bei Wohngebäuden

Im Fall von Wohngebäuden wird nur zwischen der Art des Luftwechselmechanismus (Fenster oder RLT-Anlage), aber nicht zwischen Heiz- und Kühlfall unterschieden. Die Lüftungsleitwerte L_V sind jeweils von Luftvolumenstrom ν_V abhängig.

Im Modell für Wohngebäude mit Fensterlüftung ist Tabelle 2 zum Lookup der Luftwechselrate n_L über Konnektoren mit der Gleichung zur Ermittlung von ν_V verknüpft, und somit im Modell sichtbar. (Das Modell ist im Projektverlauf relativ früh entstanden. In später entwickelten Teilmodellen wurde diese Tabelle direkt in Gleichungen eingebettet.)

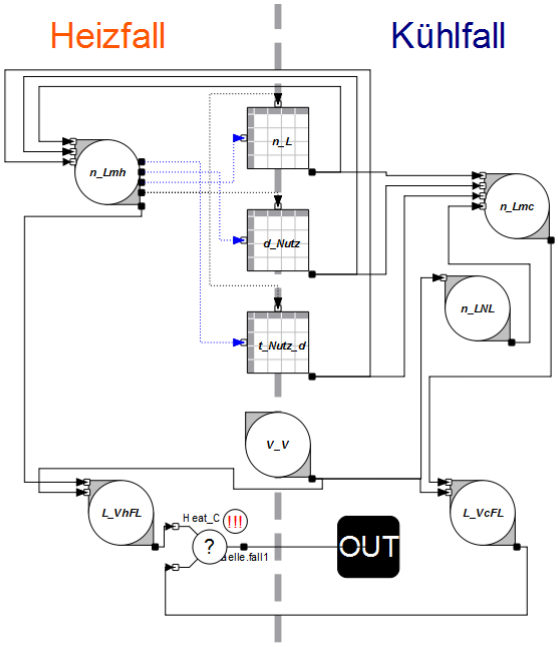


Abbildung 4.3: Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit Fensterlüftung $L_{V,FL}$

Im Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit Fensterlüftung $L_{V,FL,NWG}$ sind beide Fälle – Heizfall und Kühlfall – im selben Modell implementiert, weil die beiden Formeln ähnlich sind und viele der verwendeten Größen (nämlich jene, die in der Abbildung direkt auf der grau-strichlierten, vertikalen Trennlinie liegen) in beide Fälle eingehen. Tabelle 2 ist auch hier „sichtbar“ ins Modell integriert.

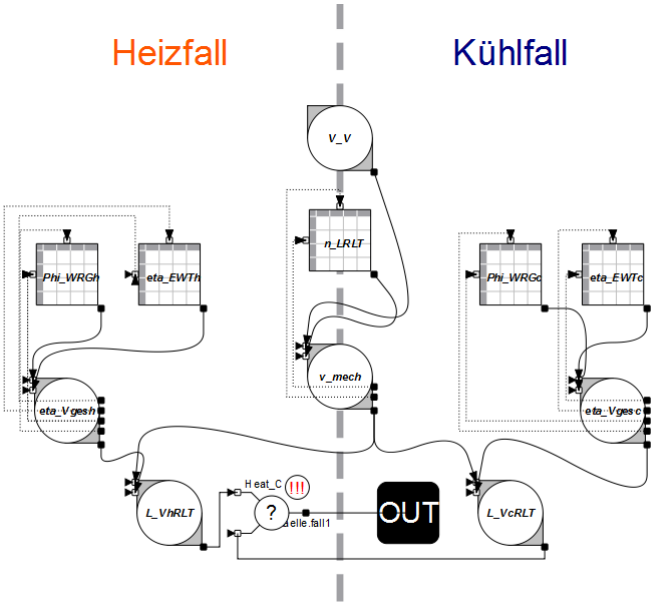


Abbildung 4.4: Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit raumlufttechnischen Anlagen $L_{V,RLT}$

Im Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit raumlufttechnischer Lüftung $L_{V,RLT,NWG}$ sind ebenfalls Heiz- und Kühlfall in *einem* Modell implementiert. Tabelle 2 wurde im Projektverlauf hier zum letzten Mal „sichtbar“ ins Modell integriert.

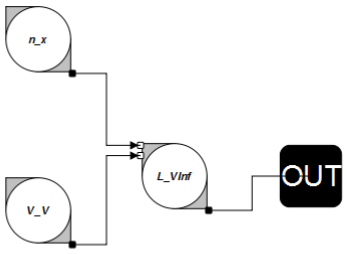
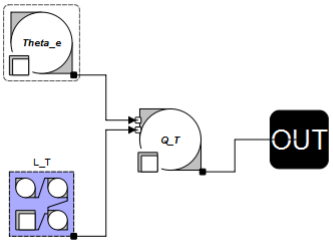


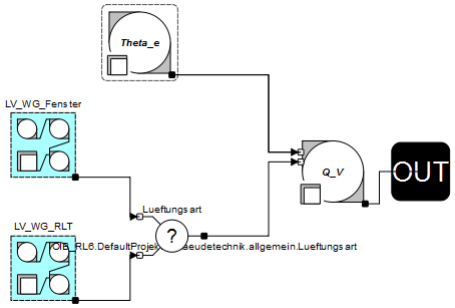
Abbildung 4.5: Modell des infiltrationsbedingten Lüftungsleitwertes bei Nichtwohngebäuden $L_{V,inf}$

Dass das „mikroskopisch kleine“ Teilmodell zur Ermittlung des infiltrationsbedingten Lüftungsleitwerts überhaupt als eigenes Modell umgesetzt wurde, anstatt es z.B. im

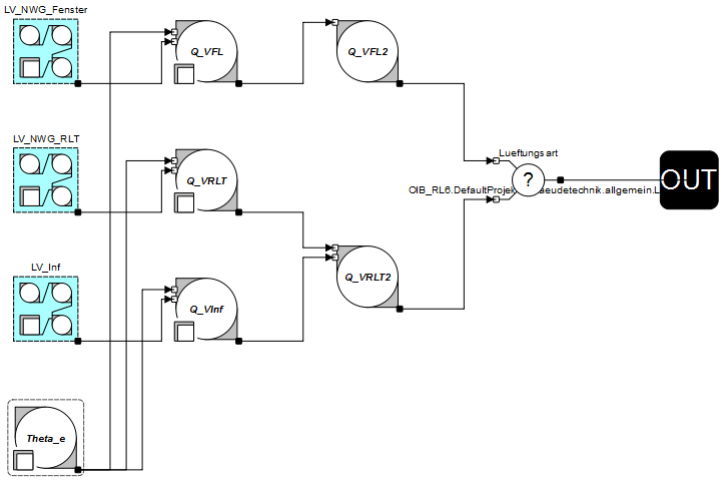
Teilmodell zur Ermittlung von Q_V zu integrieren, liegt eigentlich nur daran, dass es von der strukturellen Logik sonst nirgends wirklich dazu passt. Vielleicht macht sich de Aufwand später bezahlt, wenn es darum geht, dieses Modell detaillierter zu formulieren.



(a) Transmissionswärmeverlust Q_T bei Wohn- und Nichtwohngebäuden



(b) Lüftungswärmeverlust Q_V bei Wohngebäuden



(c) Lüftungswärmeverlust Q_V bei Nichtwohngebäuden

Abbildung 4.6: Modelle der Transmissions- und Lüftungswärmeverluste

Die Teilmodelle zur Ermittlung der Transmissions- und Lüftungsleitwerte gehen in die Teilmodelle zur Ermittlung der Transmissions- und Lüftungswärmeverluste ein.

Die Außentemperatur Θ_e ist in den Modellen als `replaceable` deklariert und wird deshalb mit einem strichlierten Rahmen dargestellt. Die Redeklarierbarkeit von Θ_e ist naheliegend, weil alleine in der ÖNORM B 8110-6 (2010) schon zwei verschiedene

Temperaturmodelle vorhanden sind, und im Rahmen dieser Arbeit auch eine Schnittstelle zu Daten aus Testreferenzjahren geschaffen wurde, d.h., das Temperaturmodell kann auf mehrere Arten variieren.

Bei den Lüftungsleitwerten erfolgt hier die Fallunterscheidung zwischen Fensterlüftung und RLT-Lüftung, und es wird nur der relevante Wert weiter gegeben. Dass dies in der „Rechenkette“ erst hier passiert, hat den Grund, dass es später prinzipiell möglich sein sollte, Modelle für Fensterlüftung und mechanische Lüftung zu kombinieren – dort, wo jetzt die Fallunterscheidung stattfindet, wäre dann der logische Ansatzpunkt.

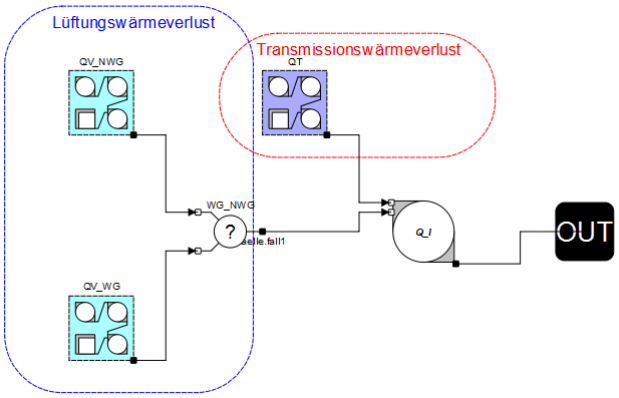
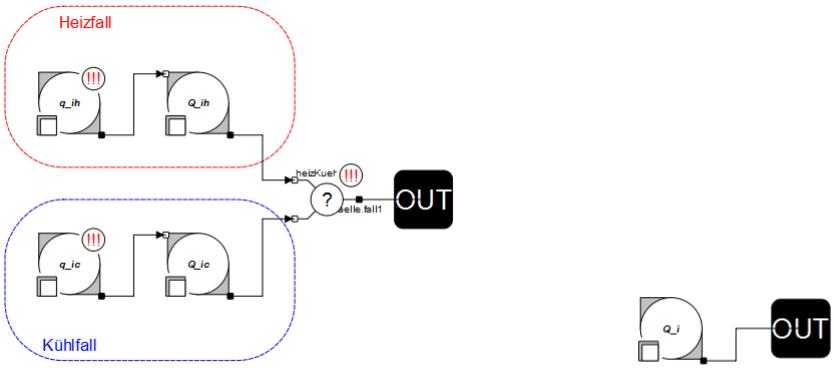


Abbildung 4.7: Modell der Gesamtwärmeverluste Q_L

Im Teilmodell zur Ermittlung der Gesamtwärmeverluste werden schließlich die Transmissions- und Lüftungswärmeverluste summiert.

4.2.2 Wärmegewinne Q_g



(a) Nichtwohngebäude

(b) Wohngebäude

Abbildung 4.8: Modelle der inneren Wärmegewinne Q_I

Bei Nichtwohngebäuden wird zur Ermittlung der inneren Wärmegewinne wieder zwischen Heiz- und Kühlfall unterscheiden, während bei Wohngebäuden diese Unterscheidung nicht stattfindet.

In diesem Modell wurden alle vorkommenden Tabellen-Lookups direkt in die jeweiligen Formeln integriert. Um diese „unsichtbaren“ Tabellen trotzdem visuell sichtbar zu machen, wurde das Icon-Overlay in Form einer kleinen Tabelle links unten im Symbol der betreffenden Gleichungen eingeführt.

Dass die inneren Wärmegewinne von Wohngebäuden als eigenes Teilmodell implementiert wurden, obwohl es sich eigentlich um einen einzigen trivialen Rechenschritt handelt, hat dieselben Gründe, wie beim Modell zur Ermittlung des infiltrationsbedingten Lüftungsleitwertes.

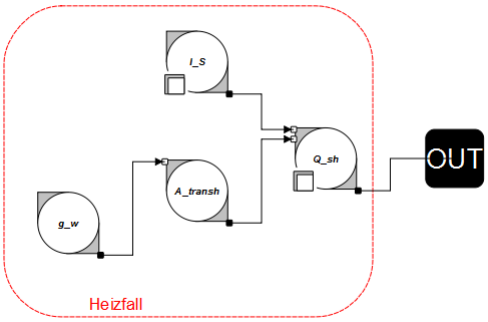


Abbildung 4.9: Modelle der solaren Wärmegewinne Q_S bei Wohngebäuden

Bei Wohngebäuden werden solare Gewinne nur im Heizfall ermittelt. Auch in diesem Teilmodell sind alle vorkommenden Tabellen direkt in die Gleichungen integriert. Das „!!!“-Icon-Overlay bedeutet in dem Fall nur, dass der Beleuchtungsenergiebedarf LENI in dem Modell (noch) statisch programmiert ist, weil die ÖNORM H 5059 im Rahmen des Projektes noch nicht fertig implementiert ist.

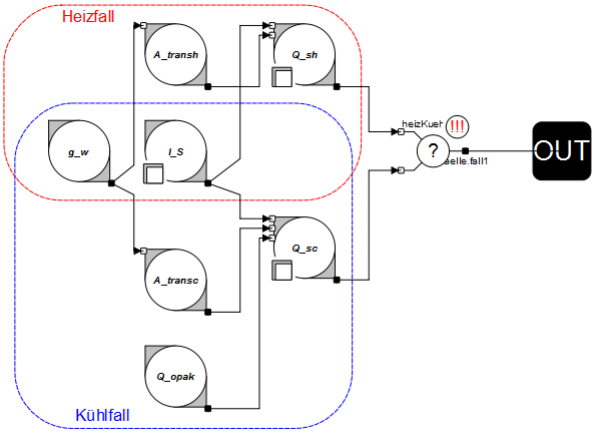


Abbildung 4.10: Modelle der solaren Wärmegewinne Q_S bei Nichtwohngebäuden

Bei Nichtwohngebäuden werden solare Gewinne nur für Heiz- und Kühlfall ermittelt, und auch hier sind alle vorkommenden Tabellen direkt in die Gleichungen integriert.

4.3 Numerisches Anwendungsbeispiel

Die in ➪ Abschnitt 4.1 ff beschriebenen Norm-Rechenmodelle sowie die Idee einer Kopplung dieser Rechenverfahren an Elemente dynamischer Gebäudesimulation wurden anhand eines exemplarischen Mehrzonenmodells erprobt.

Das Gebäudemodell gliedert sich in eine konditionierte Zone und einen unkonditionierten Pufferraum. Die konditionierte Zone wird dabei im Monatsbilanzverfahren unter Anwendung der Methoden der ÖNORM B 8810 abgebildet. Der Pufferraum wird hingegen als voll dynamisches System mit temperaturabhängig gesteuerter Gebäudetechnik und eigener Wärmekapazität abgebildet.

Als Außentemperaturen und Globalstrahlungs-Intensitäten wurden zeitdiskrete Stunden-Mittelwerte aus dem Testreferenzjahr für Wien (Innere Stadt) verwendet. Im (dynamisch simulierten) Teilmodell des unkonditionierten Pufferraumes wurden diese Werte direkt verwendet. Zur Demonstration der Umsetzung hybrider Zeitgitter unter Verwendung der entwickelten Simulationsumgebung wurden die Temperatur- und Strahlungswerte monatsweise gemittelt, und diese Monatsmittelwerte wurden dem (vereinfacht berechneten) Teilmodell der konditionierten Zone zugrunde gelegt[1].

Das exemplarische System repräsentiert also die Idee einer Simulation mit Kopplung von dynamischer Gebäudesimulation an vereinfachte, stationäre Rechenverfahren (hybrides Modell) mit unterschiedlichen Zeitschrittweiten (hybrides Zeitgitter) vollständig.

Im Folgenden werden die Modelle der beiden thermischen Zonen des exemplarischen Gebäudemodells sowie die Umsetzung des hybriden Zeitgitters im Detail beschrieben. In ➪ Abschnitt 4.3.3 werden dann einige Ergebnisse der exemplarischen Berechnung gezeigt.

1 Das Beispiel dient zur Demonstration der prinzipiellen Vorgangsweise in Fällen, in denen ein Teilmodell am Monats-Zeitgitter *nicht* auf ein Stunden-Zeitgitter übertragen werden soll/kann. Im konkreten Fall wäre (wegen der stationären Formulierung der Norm-Rechenmethoden) eine Übertragung des kompletten Modells vom 12-Monats- aus 8760-Stunden Zeitgitter problemlos möglich, und das Endergebnis der stundenweisen Bilanzierung mit Stundenwerten wäre natürlich das selbe, wie bei der monatsweisen Bilanzierung mit Monatsmittelwerten.

4.3.1 Details des exemplarischen Gebäudemodells

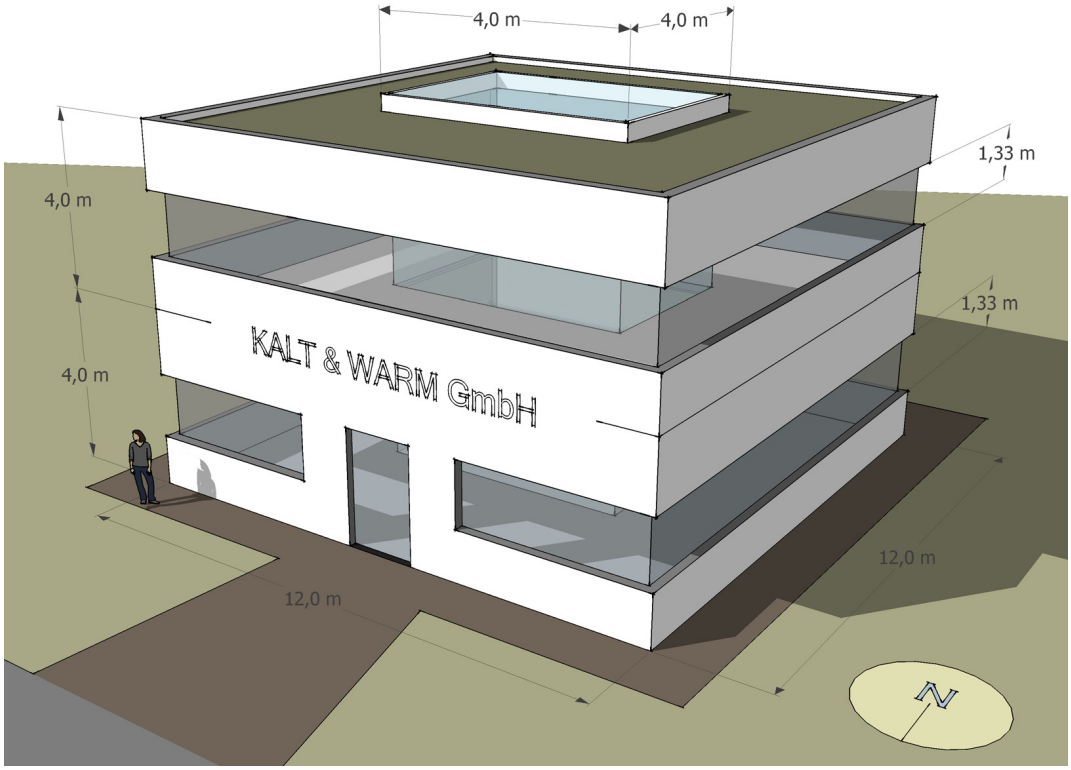


Abbildung 4.11: Visualisierung des Gebäudemodells

In ➪ Abbildung 4.11 ist das Gebäude skizziert, welches dem numerischen Anwendungsbeispiel zugrunde liegt: Eine zweigeschoßige, konditionierte Zone umschließt ein unkonditioniertes Treppenhaus, dessen Glasdach an den Außenraum grenzt.

4.3.1.1 Der unkonditionierte Pufferraum

Wie schon erwähnt, ist der Pufferraum als dynamisch simuliertes Teilmodell am 8760-Stunden-Zeitgitter abgebildt. Die ➪ Abbildungen 4.12 und 4.13 dienen der Veranschaulichung das thermischen Verhaltens des Raumes sowie der gebäudetechnischen Steuerung, und sind unabhängig vom exemplarischen Gebäudemodell. Sie zeigen verschiedene Parameter des Pufferraumes während der ersten 72 Stunden eines

Einschwingvorganges bei periodischen Randbedingungen[1] :

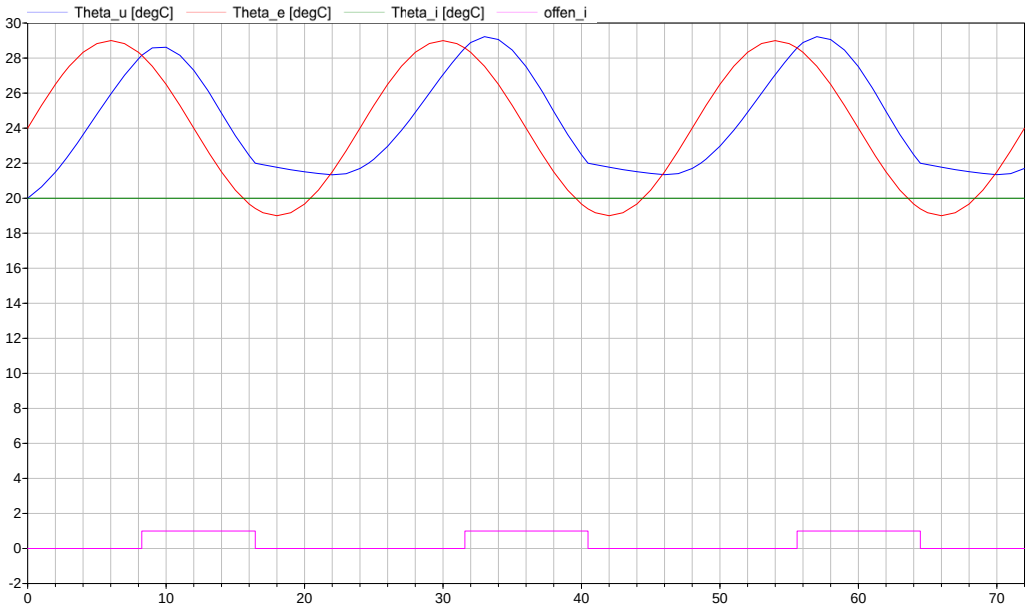


Abbildung 4.12: Temperaturverläufe während eines Einschwingvorganges: blau: Pufferraumtemperatur Θ_u, rot: Außentemperatur Θ_e, grün: Temperatur der konditionierten Zone Θ_i, magenta: Öffnungszustand der Lüftungsklappe

Dem abgebildeten Verhalten des Pufferraumes liegen folgende Parameter zugrunde:

- Glasdach: $U = 1{,}60\ \frac{W}{m^2 \cdot K}$, $g = 0{,}35$, Rahmenanteil: 20%
- Sonnenschutzeinrichtung: keine
- Trennwand zur konditionierten Zone: $U = 0{,}50\ \frac{W}{m^2 \cdot K}$
- Schalttemperaturen zum Öffnen / Schließen der Lüftungsklappe: 26 °C / 22 °C
- Luftwechsel bei geöffneter / geschlossener Lüftungsklappe: 10,0 / 0,10
- Wärmekapazität des Pufferraumes: $1{,}3\dot{3}\ \frac{W \cdot h}{K}$
- Geometrie und Orientierung lt. ➪ Abbildung 4.11

Zum Öffnen der Lüftungsklappe muss neben Erreichen der Schalttemperatur auch

1 Das Modell dieses „Puferraum-Tests“ ist im der Objekthierarchie der etwickelten Umgebung unter *OIB_RL6.pufferRaum_test* zu finden und kann autonom simuliert werden.

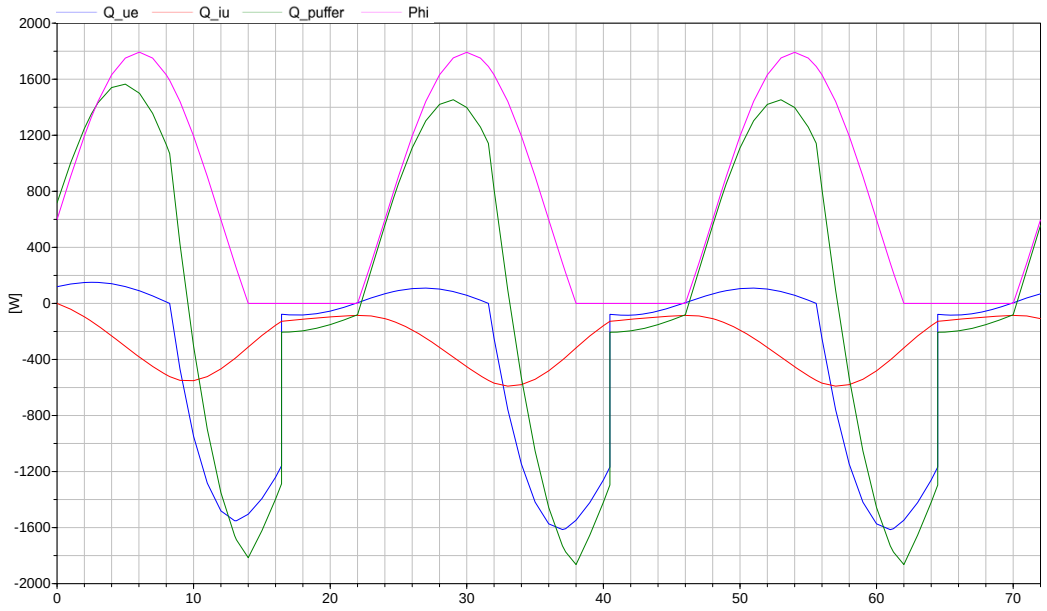


Abbildung 4.13: Wärmeströme während eines Einschwingvorganges: blau: Wärmestrom Q_{ue} von außen in den Pufferraum, rot: Wärmestrom Q_{iu} von der konditionierten Zone in den Pufferraum, grün: Wärmestrom Q in die Speicherkapazität des Pufferraumes, magenta: solare Wärmegewinne Φ durch das Glasdach

die Bedingung Erfüllt sein, dass die Pufferraumtemperatur Θ_u über der Aussenlufttemperatur Θ_e und über der Temperatur der konditionierten Zone Θ_i liegt.

Der Wärmestrom zwischen Pufferraum und Erdreich bleibt vernachlässigt, weil die Grundfläche des Pufferraumes rundum von der konditionierten Grundfläche umgeben ist. Der Luftwechsel zwischen der konditionierten Zone und dem unkonditionierten Pufferraum wird gemäß EN ISO 13789, Formel (10) ebenfalls mit null angesetzt.

Die aufgelisteten Parameter sind auch der Berechnung des exemplarischen Gebäudemodells zugrunde gelegen.

☞ **Hinweis:** Die Wärmekapazitäten der raumbegrenzenden Flächen und der Objekte im Pufferraum sind in Form eines pauschalen Wertes abgebildet. Wird dieser Wert auf null gesetzt, und wird die Luftwechselzahl n_{ue} bei geöffneter und geschlossener Lüftungsklappe auf den selben Wert gesetzt, so entspricht das Modell des Pufferraumes genau dem eines unkonditionierten Pufferraumes gemäß EN ISO 13789, d.h. das

das detaillierte Verfahren nach ÖNORM B 8110-6 für unkonditionierte Gebäudeteile kann mit dem entwickelten Pufferraum-Modell abgebildet werden.

4.3.1.2 Die konditionierte Zone

Wie schon erwähnt, ist die konditionierte Gebäudezone als nach ÖNORM B 8110 berechnetes Teilmodell am 12-Stunden-Zeitgitter abgebildt. Grundlage der Implementation sind die in ➪ Abschnitt 4.1 ff vorgestellten Teilmodelle. Die Bauweise und Gebäudeausstattung der Zone wurden wie folgt angenommen:

- Mittelschwere Bauweise, weiße Wände, graues Dach, keine Unterkellerung
- $\frac{1}{3}$ Fensterflächenanteil in allen Außenwänden
- Sonnenschutzeinrichtung: Innenjalousie ($z = 0{,}78$), manuell betätigt
- Gesamtenergiedurchlassgrad der Verglasungen: $g = 40\%$
- Geometriebedingte Verschattungsfaktoren (pauschal): $F_{S,c} = 1{,}00$, $F_{S,h} = 0{,}75$
- Raumlufttechnik-Anlage mit Kreuzstromwärmetauscher (Wärmebereitstellungsgrad: 50%)
- Wärmedurchgangskoeffizienten: $U_{opak} = 0{,}20$ und $U_{transparent} = 1{,}00\ \frac{W}{m^2 \cdot K}$
- Geometrie und Orientierung lt. ➪ Abbildung 4.11
- Wärmebrücken: pauschaler Leitwertzuschlag (10%)

Da die Normen zu deren Ermittlung sind nicht Gegenstand dieser Arbeit sind, wurde der Beleuchtungsenergiebedarf gemäß ÖNORM H 5059 wurde mit $LENI = 30{,}0\ \frac{kW \cdot h}{m^2 \cdot a}$ und die Heizgrenztemperatur gemäß ÖNORM H 5056 mit 12,0 °C gewählt gewählt.

4.3.2 Details zur Konvertierung zwischen Stunden- und Monatszeitgitter

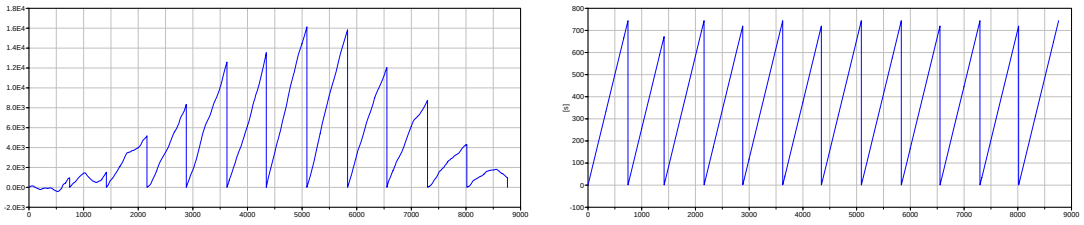
In ➪ Abschnitt 3.1.1 wurden verschiedene Möglichkeiten gegenseitiger Abhängigkeit von Größen am Monats- und am Stundengitter bei Simulationen mit hybridem Zeitgitter theoretisch behandelt. Das im Folgenden beschriebene Verfahren entspricht dem Fall gemäß ➪ Abschnitt 3.1.1.1, bei dem Größen am groben Zeitgitter abhängig von Größen am feinen Zeitgitter sind.

Der einfachere, umgekehrte Fall (Größen am feinen Zeitgitter sind abhängig von Größen am groben Zeitgitter; ➪ Abschnitt 3.1.1.2) ist eigentlich selbsterklärend: Hier ist keine Konvertierung nötig, sondern die Werte stehen jederzeit auf beiden Zeitgittern zur Verfügung. Zirkelbeziehungen zwischen Größen auf verschiedenen

Zeitgittern können ohne zusätzliche Software nicht behandelt werden, und sind wie in ➪ Abschnitt 3.1.1.3 beschrieben aufzulösen.

Im Folgenden wird am Beispiel der Außentemperatur aus dem Testreferenzjahr für die Wiener Innenstadt (➪ siehe Abbildung 2.4) detailliert dargestellt, wie eine Größe am 8760-Stunden-Zeitgitter mit Hilfe der entwickelten Software monatsweise gemittelt wird, sodass am Ende eines Monats jene Größen am 12-Monats-Zeitgitter, die von diesem Mittelwert abhängig sind, berechnet werden können.

Zuerst werden ein Temperaturintegral und eine Zeitfunktion gebildet, die am Ende jedes Monats resettet werden. Dies passiert mit Hilfe der Funktion *OIB_RL6. Funktionen. timeConversion. totalHoursPerMonth*, die angibt, bei welcher Simulationszeit (`time`) „die letzte Stunde eines bestimmten Monats geschlagen hat“. Die Funktion lässt sich einerseits dafür verwenden, um am Ende jedes Monats ein Event zu triggern, bei dem das Temperaturintegral mittels `reinit` auf null zurückgesetzt wird (➪ siehe Abbildung 4.14a). Andererseits lassen sich mit dieser Funktion „lokale Zeitlinien“ einführen, die am Anfang jedes Monats beginnen, indem das Ende des Vormonats von der aktuellen Simulationszeit abgezogen wird (➪ siehe Abbildung 4.14b).



(a) monatliches Temperaturintegral **(b)** monatliche Zeitfunktion

Abbildung 4.14: Eingangswerte zur Ermittlung der Monatsmitteltemperatur

Dividiert man die so erhaltene Funktion des Temperaturintegrals durch jene des Monatsalters, so erhält man die Mitteltemperatur – gemessen jeweils vom Beginn des Monats (➪ siehe Abbildung 4.15).

Die einzigen Werte, die in ➪ Abbildung 4.15 wirklich von Interesse sind, sind die Monatsmitteltemperaturen – also die Werte am Ende der Monate. Diese werden am Ende jedes Monats von den temperaturabhängigen Größen im Teilmodell am 12-Monats-Zeitgitter übernommen, und dort weiter verwendet. Die in ➪ Abbildung 4.15 dargestellte Funktion ist bereits geeignet, um den am Monatsende den Wert

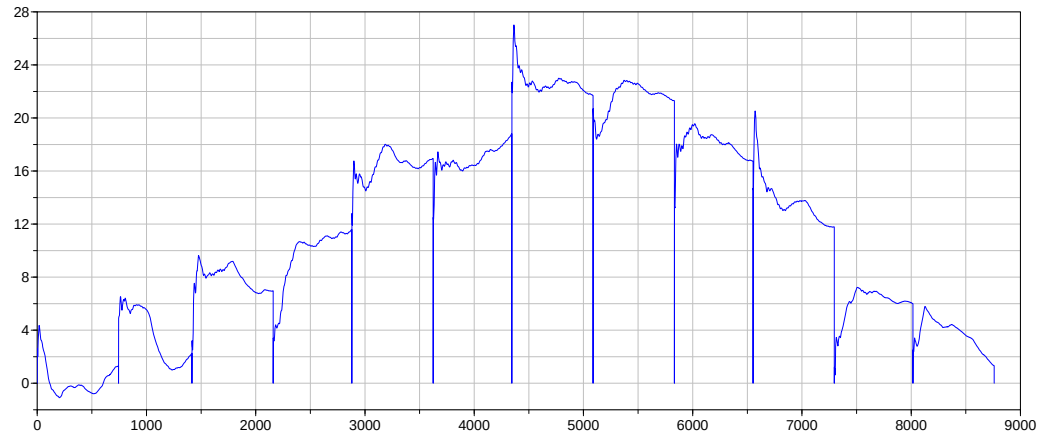
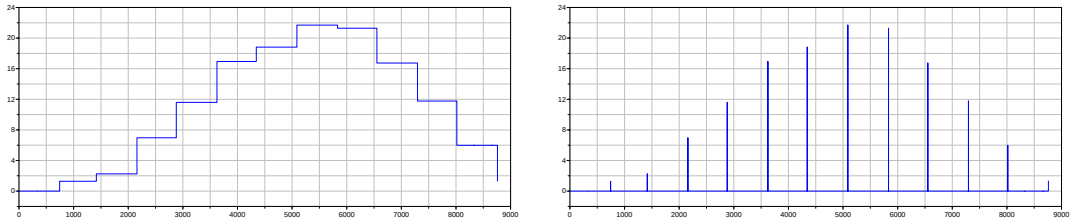


Abbildung 4.15: monatliche Mitteltemperatur

der Monatsmitteltemperatur bereitzustellen. Unter Umständen kann es sinnvoll sein, die Funktion noch ein wenig zu „frisieren“, sodass keine irrelevanten Werte mehr vorkommen ➪ Abbildungen 4.16b und 4.16a.

Die gesamte hier vorgestellte Methodik ist in der Basisklasse für Testreferenzjahre *OIB_RL6.Klima.TRY.Prototyp* implementiert und umfasst nur wenige Programmzeilen. Die Methode lässt sich auf beliebige Größen am 8760-Stunden-Zeitgitter anwenden. Im konkreten Fall des exemplarischen Gebäudemodells wird z.B. auch der Wärmestrom Q_{iu} auf diese Art in einen Monatsmittelwert umgerechnet.



(a) ohne Reinitialisierung auf null **(b)** mit Reinitialisierung auf null

Abbildung 4.16: Monatsmitteltemperatur

4.3.3 Zusammenführung in ein hybrides Modell

Sobald alle Kopplungsgrößen auf beiden Zeitgittern verfügbar sind, können die beiden Teilmodelle „dynamisch simulierter Pufferraum“ und „nach Norm berechnete, konditionierte Zone“ in ein gekoppeltes System zusammen geführt und simuliert werden. Exemplarisch wurden die Transmissionswärmeverluste des gekoppelten Gesamtsystems für die Dauer eines Jahreszyklus ermittelt und ausgewertet.

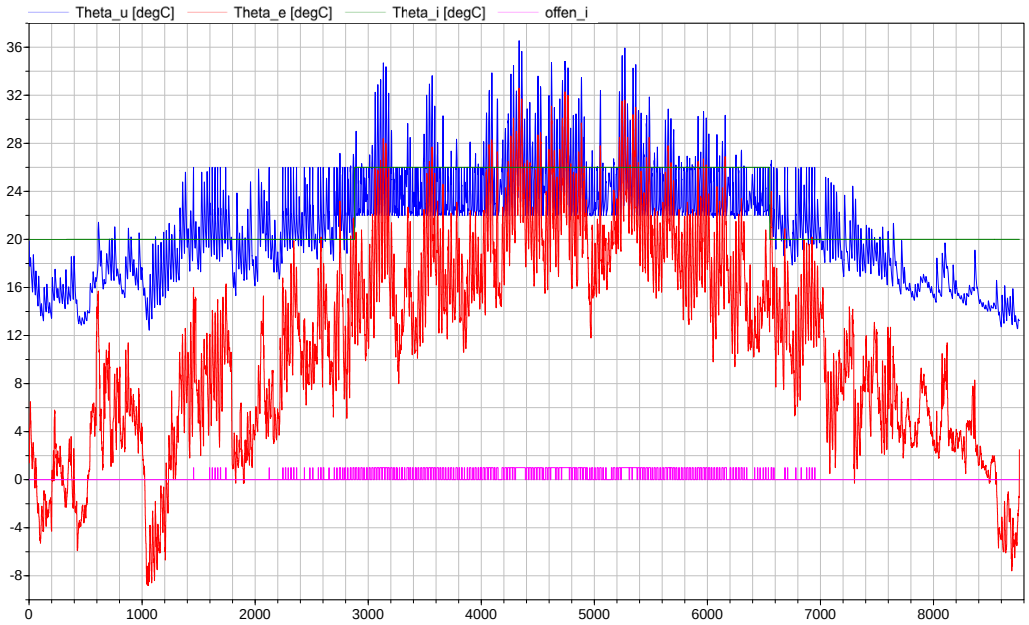


Abbildung 4.17: Jahrestemperaturverläufe: blau: Pufferraumtemperatur Θ_u, rot: Außentemperatur Θ_e, grün: Temperatur der konditionierten Zone Θ_i, magenta: Öffnungszustand der Lüftungsklappe

➪ Abbildung 4.17 zeigt die Jahresverläufe von Außentemperatur Θ_e, Pufferraumtemperatur Θ_u und Temperatur im konditionierten Bereich Θ_i, sowie den Öffnungszustand der Lüftungsklappe des Pufferraumes. Es ist unschwer zu erkennen, welche Größen auf welchem Zeitgitter liegen. Außerdem sieht man schön, dass bei den angenommenen Parametern (Speichermasse, Luftwechsel, Lüftungsklappen-Steuerung, etc.) eine Überhitzung des Stiegenhauses während der Sommermonate nicht verhindert wird.

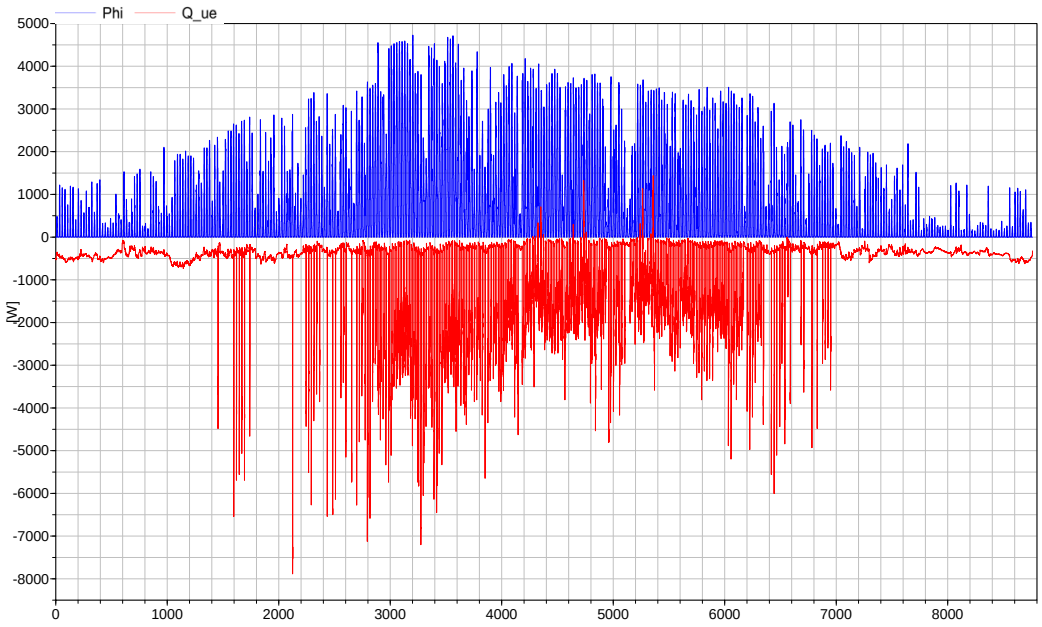


Abbildung 4.18: Jahresverlauf der Wärmeströme von außen in den Pufferraum: blau: solare Wärmegewinne, rot: Transmissionswärmestrom

In ➪ Abbildung 4.18 werden die Energieeinträge in den Pufferraum dargestellt. Der Transmissionswärmestrom Q_{ue} ist praktisch immer negativ, d.h., dass die Pufferraumtemperatur Θ_u fast ständig über der Außentemperatur Θ_e liegt.

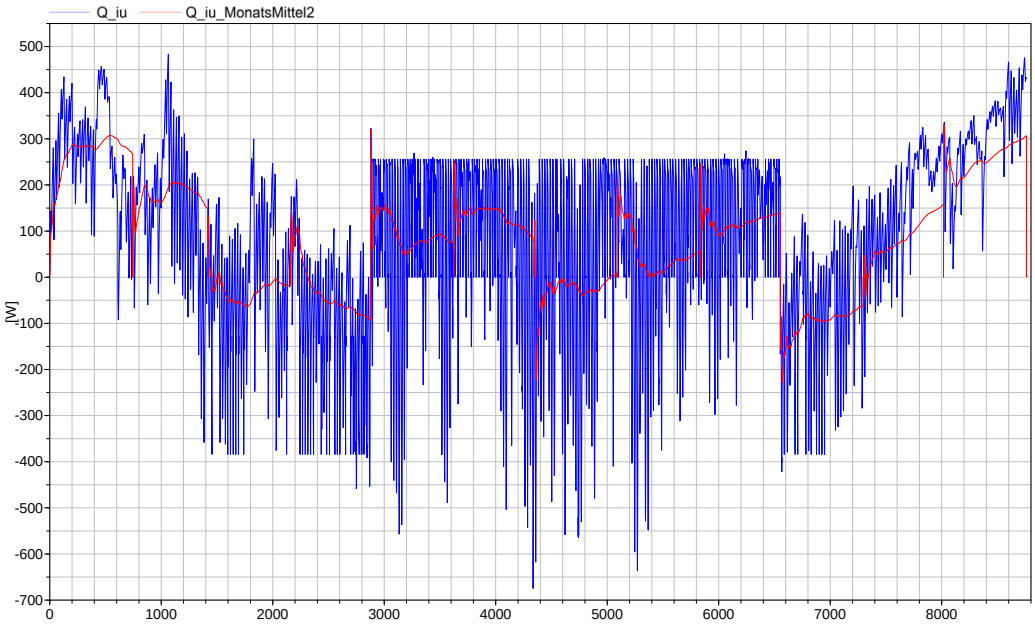


Abbildung 4.19: Jahresverlauf des Wärmestroms von der konditionierten Zone in den Pufferraum: blau: Wärmestrom, rot: monatlich berechneter Mittelwert

Im Wärmestrom vom konditionierten Bereich in den Pufferraum spiegelt sich der plötzliche Sprung der Temperatur Θ_i im konditionierten Bereich beim Übergang von der Heizperiode auf die Kühlperiode wider. Die in ➪ Abbildung 4.19 dargestellten monatlichen Temperaturmittelwerte sind nur an den Monatsübergängen relevant (➪ siehe Abschnitt 4.3.2), und finden sich auch in ➪ Abbildung 4.20 wieder.

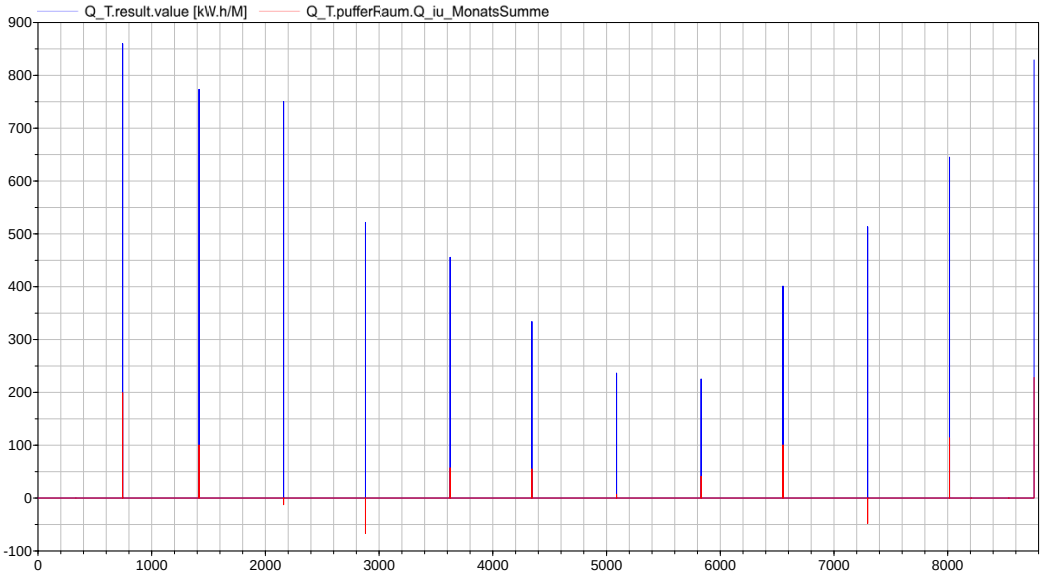


Abbildung 4.20: Transmissionswärmeverluste der konditionierten Zone (Monatsbilanzverfahren): blau: Gesamt-Transmissionswärmeverluste Q_T, rot: Transmissionswärmeverluste zum Pufferraum

Den zwölf Monatsbilanzwerten der Transmissionswärmeverluste liegen die Monatsmittelwerte der Außentemperatur Θ_e und der Transmissionswärmeverluste in den Pufferraum Q_u zugrunde, welche mit den in ➪ Abschnitt 4.3.2 beschriebenen Methoden aus dem dynamisch simulierten Teilmodell ermittelt wurden. Alle anderen Eingangswerte in die Berechnung der Transmissionswärmebilanz (Nutzungsprofil, Transmissionsleitwert, etc.) wurden nach den Norm-Rechenverfahren der ÖNORM B 8110 ermittelt.

A Hinweise zur Entwicklung mit Modelica

A.1 Allgemeines

So weit der Autor in Erfahrung bringen konnte, existiert keine vollständige Implementation der Modelica Programmiersprache entsprechend der Modelica Language Specifications 3.x [Mod07] [Mod09] [Mod10]. Die verschiedenen Hersteller beschränken sich jeweils auf die Implementation von Subsets des Language Standard, welche auf die Handhabung der Modelica Standard Library MSL[1] bzw. proprietärer Bibliotheken spezialisiert sind.

Es ist nichts Ungewöhnliches, dass Implementationen von Programmiersprachen ihre Eigenheiten haben, aber im Fall von Modelica wird dieser Umstand zum Problem, weil die Hersteller nicht genau dokumentieren, welches Subset des offiziellen Language Standard von ihren Tools unterstützt wird. Werden nun eigene Bibliotheken (mangels Hersteller-Dokumentation) unter Verwendung der Modelica Language Specification entwickelt, so stößt man zwangsläufig an die Grenzen der jeweiligen Modelica-Implementation. So gerät die Programmierung anfänglich zur „reinen Glückssache“ und erst durch zwangsläufig auftretende Probleme erfährt der Programmierer mehr über die Stärken und Schwächen seiner gewählten Modelica-Umgebung.

Zitate aus entsprechenden Supportanfragen:

> We do not have a complete description of what is supported and not. The current status is in principle what is covered by the subset of Modelica standard library that is shipped with MathModelica. (Email *MathCore* 19.04.2010 10:33)

> We implemented mostly what is needed to flatten the libraries. We'll get to it [Anm.: gemeint ist hier eine vollständige Implementation des `enumeration`-Typs] after we are able to flatten MSL 3.x MultiBody and Media libraries which are

1 siehe ➪ `http://www.modelica.org/libraries/Modelica`

> on highest priority right now. MultiBody is almost ready and Media is on the way. (Adrian Pop am 14.05.2010 00:58 im *OpenModelica* Forum)

☞ **Hinweis:** Im gegenständlichen Projekt wurde anfänglich auf *OpenModelica* zur Programmierung und auf *MathModelica* zur Modellierung gesetzt. Angestrebt war, dass das entwickelte Framework in beiden Umgebungen lauffähig ist. Im Laufe der Entwicklung musste die Kompatibilität zu *OpenModelica* aufgegeben werden (➪ siehe Abschnitt A.2.1), und *MathModelica* musste durch *Dymola* als Modellierungs- und Simulationsumgebung ersetzt werden.

A.2 Hinweise zu den untersuchten Modelica-Umgebungen

A.2.1 OpenModelica

Eine wesentliche Schwierigkeit bei der Ausarbeitung war, dass zum Zeitpunkt der Programmierung die letzte stabile Version von *OpenModelica* (Version 1.4.5) die Modelica 3.0 Syntax zwar bereits unterstützte, aber noch keine Modelica-3.0-typischen Sprachelemente interpretieren konnte, und somit eigentlich bereits obsolet war. Die *OpenModelica* Version 1.5 lag bereits als Release Candidate vor (anfänglich RC1, später RC2, RC3 und schließlich eine „final“ Release) und unterstützte den Sprachstandard 3.0 weitgehend. Im Sinne einer zukunftssicheren Lösung wurde die Entscheidung getroffen, die Programmierung auf Basis der Release Candidates auszuarbeiten.

Erst im Zuge der Ausarbeitung wurde dann klar, dass in diesen sog. „Release Candidates“ (entgegen der üblichen Praxis) weder alle Funktionen der endgültigen Version enthalten, noch alle bis dahin bekannten Fehler in der Software behoben waren – die verwendete Software war also de facto eine frühe Beta-Version und (entgegen der offiziellen Bezeichnung) kein Release Candidate im üblichen Sinn. Auch die sog. „final“ Release, die gegen Ende der Ausarbeitung dieser Arbeit verfügbar war, entspricht nicht den üblichen Standards von produktionsstabiler Software.

Die anfänglich angestrebte Kompatibilität der entwickelten Simulationsumgebung zur kostenlosen *OpenModelica*-Umgebung musste aufgegeben werden, als der Umfang dieser Probleme erkannt wurde (➪ siehe Abschnitt 2.2.2).

A.2.2 MathModelica & Dymola

MathModelica musste im Zuge der Ausarbeitung durch *Dymola* ersetzt werden. Die Gründe dafür sind ähnlich wie jene, die zur Aufgabe der Kompatibilität mit *OpenModelica* gezwungen haben (➪ siehe Abschnitt 2.2.3.1).

A.3 Konkrete Bugs & fehlende v3.0-Features in den untersuchten Modelica-Umgebungen

Im Folgenden werden einige bis jetzt *undokumentierte* Probleme aufgelistet, die im Zuge des Projektes erkannt wurden. Es sind hier nur jene Fehler und Probleme aufgelistet, bei denen die Ursache eingegrenzt werden konnte und die von den Anbietern der jeweils betroffenen Modelica-Plattform bestätigt sind. Die Liste wurde vor dem Hintergrund weiterer bevorstehender Modelica-Programmierungen im Forschungsbereich für Bauphysik und Schallschutz an der TU Wien in diese Arbeit aufgenommen, und soll dazu beitragen, dass die Bearbeiter dieser zukünftigen Projekte nicht erst die selben Bugs & Missing Features eruieren müssen, bevor die Entwicklung einigermaßen „flüssig von der Hand geht“.

1. *OpenModelica*, *MathModelica*: Der in Modelica 3.0 eingeführte `enumeration`-Datentyp bietet sich als ideales Konstrukt zur Indexierung von Arrays für Tabellendaten an, weil aus jedem Enumerations-Element sowohl ein (Integer-)Index wie auch ein String generiert werden kann. Somit ließe sich z.B. die Kopfzeile einer Tabelle ideal abbilden: Der String könnte den Text aus der Kopfzeile enthalten, und der Integer-Wert könnte zum Indexieren im Array der Tabellendaten verwendet werden. Gemäß [Mod07], Kap. 10.5.1 sollte ein `enumeration`-Element zur Indexierung von Arrays herangezogen werden können (statt Integer). Leider war der Enumerations-Typ in keiner der untersuchten Umgebungen zum Zeitpunkt der Erstellung dieser Arbeit vollständig implementiert, und es ist in der Programmierung bei der Umsetzung dieser Idee zu zahlreichen verschiedenen Problemen gekommen.

   Diskussion: ➪ `http://www.openmodelica.org/index.php/forum/topic?id=43`

2. *Dymola*: Die Funktion Integer(enumType.enumElement) entsprechend [Mod07] Kap. 4.8.5.2 funktioniert nicht. Entgegen dem gültigen Sprachstandard wird aber der Type-Casting Operator integer() für die Anwendung auf Enumerations-Elemente unterstützt. Dadurch entsteht das massive Problem, dass die Kompatibilität zu standardkonformen Modelica-Implementationen bei Verwendung dieses Sprachelementes gebrochen werden muss.

   Zitat aus der entsprechenden Supportanfrage:

   > Unfortunately, the conversion function Integer() is still not supported in Dymola and you should therefore use the integer() function instead. The same situation applies to the String() conversion function. It is however possible to use the realString() function instead. (Email *Dynasim* 08.09.2010 16:44)

```
1 ...
2 type myEnum = enumeration(eins "1. Element", zwei "2. Element");
3 Integer myInt = Integer(myEnum.eins); // funktioniert nicht
4 ...
```

Listing A.1: Problem

```
1 ...
2 type myEnum = enumeration(eins "1. Element", zwei "2. Element");
3 Integer myInt = integer(myEnum.eins); // nicht standardkonform!
4 ...
```

Listing A.2: Workaround

3. *OpenModelica*: Die Funktion *Integer(enumType.enumElement)* entsprechend [Mod07] Kap. 4.8.5.2 funktioniert unter gewissen Umständen nicht, wenn enumType im selben Scope deklariert ist, in dem der Funktionsaufruf stattfindet.

4. *MathModelica* 2.1.0: Probleme im Handling von redeklarierbaren Elementen führen dazu, dass deren graphische Symbole im GUI nicht auswählbar sind.

   Zitat aus der entsprechenden Supportanfrage: „The problem in the Model editor is because we didn't have full support for replaceable components in v2.1, this has been fixed for next release." (Email *MathCore* 14.04.2010 12:54)

5. *OpenModelica* 1.5 RC1/2: Bei Arrays vom Typ Boolean kommt es zu verschiedenen Problemen. In *OpenModelica* werden bei Übergabe solcher Arrays

an Funktion alle Elemente mit einem Index größer als 1 fälschlich auf `false` gesetzt.

Diskussion: ➪ http://www.openmodelica.org/index.php/forum/topic?id=27

6. *MathModelica* 2.1.2: Bei Verwendung von Arrays vom Typ Boolean kommt es zu nicht näher bekannten Problemen im Kernel, die eine Übersetzung des Modells verhindern.

   Zitat aus der entsprechenden Supportanfrage:

   > The problem was with arrays of Booleans, and since you use that quite extensively I can't propose a simple workaround. We could however make this bugfix available to you in an alpha release quite soon if you are interested (Email *MathCore* 14.04.2010 12:54)

7. *OpenModelica* 1.5 RC1/2/3: Das Adressieren von Array-Elementen entsprechend [Mod07] Kap. 10.5 funktioniert nur mittels *expliziter Werte*. Werden *Rückgabewerte* aus Funktionsaufrufen bei der Deklaration von Konstanten verwendet, so kann der generierte C++-Code u. U. nicht kompiliert werden.

```
1 class myClass
2   constant Integer myConst = integer(sin(45));
3   constant Real myArray[2] = {2.1, 3.2};
4   Real y;
5   equation
6     y = myArray[myConst];
7 end myClass;
```

Listing A.3: Problem

8. *OpenModelica* 1.5 RC1/2/3: Wenn zur Konstruktion von konstanten Arrays Rückgabewerte aus Array-Funktionen verwendet werden, so kommt es zu Fehlern bei der Übersetzung des generierten C++-Codes.

   Diskussion: ➪ http://openmodelica.org/index.php/forum/topic?id=42

```
1 class defect
2   constant Real i[2,2] = transpose({{1.1,2.2},{3.3,4.4}});
3   Real r;
4   equation
5     r = i[1,2];
```

```
6 end defect;
```

Listing A.4: Problem

```
1 class working
2   constant Real i[2,2] = {{1.1,2.2},{3.3,4.4}};
3   Real r;
4   equation
5     r = i[2,1]; // Vertauschen der Indizes statt Transponieren
          der Matrix
6 end working;
```

Listing A.5: Workaround

9. *OpenModelica MathModelica*: Die Verwendung von Quoted Identifiers (Q-IDENT Schreibweise gem. [Mod07] Kap. 2.3.1) führt bei Verwendung von Leerzeichen oder Sonderzeichen im Bezeichner zu nicht kompilierbarem C++-Code. Das Problem tritt insbes. mit den von *MathModelica* automatisch generierten Bezeichnern von Objekten auf, deren Klassen einen solchen Namen haben.

   Zitat aus der entsprechenden Supportanfrage: „Quoted identifiers might be an issue (I have to check if we escape them somehow before we reach C code).“ (OpenModelica-Forum 09.10.2010 03:23)

   Diskussion: ➪ http://www.openmodelica.org/index.php/forum/topic?id=91&p=1#p562

```
1 class '(1) Number One'
2   ...
3 end '(1) Number One';
4
5 model myModel
6   ...
7   '(1) Number One' '(1) Number One Instanz';
8   ...
9 end myModel;
```

Listing A.6: Problem

```
1 class Number_One
2   ...
3 end Number_One;
4
5 model myModel
```

```
  ...
  Number_One Number_One_Instanz;
  ...
end myModel;
```

Listing A.7: Workaround

10. *OpenModelica*: Im Eclipse MDT funktioniert der Lookup von Elementen innerhalb einer Klassenhierarchie nur dann, wenn Fully Qualified Identifier verwendet werden. Führt man den Mauszeiger über einen Identifier, so ist das praktische Popup-Fenster zur Darstellung des Sourcecodes jenes Objektes, auf das verwiesen wird, nur verfügbar, wenn der Lookup funktioniert.

```
import SI = Modelica.SIunits.;
class myClass
(...)
// Deklaration mittels "abgekürztem" Identifier:
CoefficientOfHeatTransfer uWert;
(...)
end myClass;
```

Listing A.8: Problem

```
class myClass
(...)
// Deklaration mittels Fully Qualified Identifier:
Modelica.SIunits.CoefficientOfHeatTransfer uWert;
(...)
end myClass;
```

Listing A.9: Workaround

☞ **Hinweis:** Aufgrund mehrerer Probleme im Namespace-Handling von *OpenModelica* und *MathModelica* empfiehlt es sich, grundsätzlich nur Fully Qualified Identifier zu verwenden.

11. *Dymola*: Wenn Modelica-Keywords gemäß [Mod10], Kap. 2.3.3, als Teile von von Bezeichnern für Klassennamen verwendet werden, kommt es zu Übersetzungsfehlern.

    Zitat aus der entsprechenden Supportanfrage:

    > I think that it is a bug in Dymola, and I think I have found the problem. It has problems with the word encapsulated as part of a model name, so they

have made some kind of mistake in their lookup routines (encapsulated is a keyword in Modelica that restricts lookup). (Email *MathCore* 09.08.2010 10:42)

```
class encapsulatedArray
  ...
end encapsulatedArray;
```

Listing A.10: Problem

```
class encArray
  ...
end encArray;
```

Listing A.11: Workaround

12. *OpenModelica*, *MathModelica*: Wenn Schlüsselwörter aus der C++ Programmiersprache als Variablennamen verwendet werden, so wird defekter C++-Code generiert und es kann kein ausführbares Simulationsprogramm kompiliert werden.

```
...
Boolean unsigned;
...
```

Listing A.12: Problem

```
...
Boolean somthing_else;
...
```

Listing A.13: Workaround

13. *OpenModelica*, *MathModelica*, *Dymola*: Eine Redeklaration von Variablen ist nur dann möglich, wenn deren Dimension unverändert bleibt. Dies steht im klaren Widerspruch [Mod07] Kap. 7.3.3 und verhindert im konkreten Fall der gegenständlichen Entwicklung z.B., dass die selbe Basisklasse für nichtvektorielle und vektorielle Gleichungen verwendet wird.

```
class parent
  replaceable Real myReal;
  ...
end parent;

class child
```

```
  extends mother(redeclare Real myReal[2]);
  ...
end child;
```

Listing A.14: Problem

```
class parent
  Real myReal;
  ...
end parent;

class child // (which is no child actually)
  Real myReal[2];
  ...
end child;
```

Listing A.15: Workaround

14. *OpenModelica*, *MathModelica*: Bei der Redeklaration von Arrays ist nicht die selbe Syntax wie bei deren ursprünglicher Deklaration möglich[1].

```
class fieldDimensionDemo
  class parent
    replaceable Real field1[2,2] = {{0,0},{0,0}}; // works
    replaceable Real[2,2] field2 = {{0,0},{0,0}}; // works
  end parent;

  class child
    extends parent(
    redeclare Real field1[2,2] = {{1,1},{1,1}}, // works
    redeclare Real[2,2] field2 = {{1,1},{1,1}}); // FAILS
  end child;
end fieldDimensionDemo;
```

Listing A.16: Problem

```
class fieldDimensionDemo
  class parent
```

1 Dies ist der einzige Punkt in der Auflistung von Problemen, der *nicht* im Widerspruch zur Modelica Language Specification [Mod07] steht, in der unter Kap. 7.2.1 die Syntax für Redeklarationen entsprechend festgelegt ist. Das Thema wird hier trotzdem erwähnt, weil hier keine Logik in der Sprachspezifikation erkennbar ist – in einer entsprechenden Support-Anfrage an *MathCore* ist man ursprünglich hier auch von einem Bug ausgegangen (➪ https://openmodelica.org:8443/cb/issue/1258). Von *Dymola* wird sowohl die korrekte, wie auch die inkorrekte, aber logische Variante der Syntax unterstützt.

```
7     replaceable Real field1[2,2] = {{0,0},{0,0}}; // works
8     replaceable Real[2,2] field2 = {{0,0},{0,0}}; // works
9   end parent;
10
11  class child
12    extends parent(
13    redeclare Real field1[2,2] = {{1,1},{1,1}}, // works
14    redeclare Real field2[2,2] = {{1,1},{1,1}}); // works
15  end child;
16 end fieldDimensionDemo;
```

Listing A.17: Workaround

15. *Dymola*: If-Expressions gem. Language Specification [Mod07], Kap. 3.6.5, funktionieren in der ersten der beiden möglichen Schreibweisen *nicht*, d.h., der (zusammen geschriebene) `elseif`-Operator wird nicht unterstützt.

```
1 ...
2 Integer i;
3 Integer sign_of_i1=if i<0 then -1 elseif i==0 then 0 else 1;
4 ...
```

Listing A.18: Problem

```
1 ...
2 Integer i;
3 Integer sign_of_i1=if i<0 then -1 else if i==0 then 0 else 1;
4 ...
```

Listing A.19: Workaround

16. *Dymola*: Linienstärken werden teilweise nicht richtig skaliert. Dadurch kommt es im Hierarchie-Baum der geladenen Klassen (links am Bildschirm) zu unkenntlichen Darstellungen, wenn Linienstärken von mehr als 0.5mm verwendet werden.

    Weiters werden Schriftattribute (Farben, fette/kursive Schrift, etc.) nicht unterstützt, d.h. derartige Formatierungen werden von Dymola nicht angezeigt.

17. *OpenModelica*, *MathModelica*: Eine Redeklaration von Enumerations-Typen in abgeleiteten Klassen ist nicht möglich.

    Zitat aus der entsprechenden Supportanfrage:

> Thank you for you report. I can confirm the issues that you have encountered. I have created tickets for them in our development system. They were given the numbers #2963 and #2964. (Email *MathCore* 20.09.2010 11:20)

Das zweite Problem, von dem in o.a. Zitat die Rede ist, ist ein Fehler beim Lookup von Identifiern, dessen genaue Ursachen und Randbedingungen dem Autor nicht bekannt sind.

A.4 Was man anders hätte machen können

Naturgemäß gibt es retrospektiv einige Punkte, die der Autor anders umsetzen würde, wenn er mit den gesammelten Erfahrungen ein zweites Mal bei null beginnen würde:

- Obwohl dies vorher kaum absehbar war, so war es sicher ein Problem, dass auf *OpenModelica* als Plattform aufgesetzt wurde (und als „logische Ergänzung“ dazu auf *MathModelica* als Modellierungsumgebung). *OpenModelica* hat zwar den Vorteil höchster Offenheit und bietet damit ideale Möglichkeiten, auf allen Ebenen „ins Geschehen einzugreifen“, aber das Timing war leider insofern ungünstig, als diese Software zum Zeitpunkt der Entwicklung der Diplomarbeit nicht stabil genug war (➪ siehe Abschnitt A.2.1). Deshalb war eine Portierung der Entwicklung auf eine andere Plattform im laufenden Projekt nötig.
- Rückblickend ist es fast ein Luxus, dass konsequent für jeden einzelnen Rechenschritt eine eigene Klasse programmiert wurde. Die Idee dahinter war, den Rechengang möglichst klar visuell darstellen zu können, und das ist auch gut gelungen (➪ siehe Abschnitt 4). Der Aufwand dafür ist aber beachtlich. Eine effizientere Methode wäre, eindeutig zusammengehörende Gleichungen jeweils in einer Klasse zusammenzufassen. Dafür spricht auch, dass die meisten Formeln – zumindest in der ÖNORM B 8110 Serie – nur an einer Stelle im Modell benötigt werden und somit nur eine Instanz der Formel im Gesamtmodell vorhanden ist.

Literaturverzeichnis

[Bun06] Bundesgesetzblatt f. d. Republik Österreich: Energieausweis-Vorlage-Gesetz - EVAG (2006)
(Zitiert auf Seite 5)

[Cre10] Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA: Creative Commons – ShareAlike 3.0 Germany Lizenz (2010), URL http://creativecommons.org/licenses/by-sa/3.0/de/
(Zitiert auf Seite 4)

[Fre07] Free Software Foundation, Inc.: GNU General Public License, Version 3 (2007), URL http://www.gnu.org/licenses/gpl-3.0.html
(Zitiert auf Seite 21)

[Hom06] Hommel, Mathias: *Parallelisierte Simulationsprozesse für virtuelles Prototyping in der Automobilindustrie*, Dissertation, Technische Universität Braunschweig (2006)
(Zitiert auf Seiten 3, 19, and 21)

[Kos06] Kossel, Roland; Tegethoff, Wilhelm; Bodmann, Michael und Lemke, Nicholas: Simulation of complex systems using Modelica and tool coupling, in: *5th International Modelica Conference*, S. 485–490
(Zitiert auf Seite 33)

[Lin10] Linköpings University, Department of Computer and Information Science: GOSMC Public License Version 1.1 (2010), URL http://www.openmodelica.org/index.php/developer/contributor-license-agreement/140
(Zitiert auf Seite 21)

[Mod07] Modelica Association: Modelica® – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification 3.0 (2007), URL http://www.modelica.org/documents/ModelicaSpec30.pdf
(Zitiert auf Seiten 17, 18, 39, 43, 75, 77, 78, 79, 80, 82, 83, and 84)

[Mod09] Modelica Association: Modelica® – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification 3.1 (2009), URL http://www.modelica.org/documents/ModelicaSpec31.pdf
(Zitiert auf Seiten 17 and 75)

[Mod10] Modelica Association: Modelica® – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification 3.2 (2010), URL http://www.modelica.org/documents/ModelicaSpec32.pdf
(Zitiert auf Seiten 16, 17, 37, 75, and 81)

[Moo08] Moosberger, Sven: Dynamische Gebäudesimulation - für jedermann?, in: *15. Schweizerisches Statusseminar „Energie- und Umweltforschung im Bauwesen“*, S. 531–537
(Zitiert auf Seite 13)

[Pöh07] Pöhn, Christian; Pech, Anton; Bednar, Thomas und Streicher, Wolfgang: *Bauphysik – Erweiterung 1: Energieeinsparung und Wärmeschutz Energieausweis - Gesamtenergieeffizienz*, Springer, 1 Aufl. (2007)
(Zitiert auf Seite 8)

[Sof09] Sofic, Mario: *Erhöhung der Anwendbarkeit vereinfachter Berechnungsverfahren zur Bestimmung des Heizwärme- und Kühlbedarfs von Gebäuden*, Dissertation, Technische Universität Wien (2009)
(Zitiert auf Seite 7)

Abbildungsverzeichnis

2.1 Kategorisierung von Berechnungs- bzw. Simulationsmodellen 6
2.2 Jahresverlauf der Monatsmitteltemperaturen für Wien (Innere Stadt) nach ÖNORM B 8110-5 (Dreischichten-Regressionsmodell) 11
2.3 Jahresverlauf der Stunden-Mitteltemperaturen für Wien (Innere Stadt) nach ÖNORM B 8110-5 (halbsynthetisches Klimadatenmodell) 12
2.4 Jahresverlauf der Stunden-Mitteltemperaturen des Testreferenzjahres für Wien (Innere Stadt) 12
2.5 Typisches Ablaufschema vom Modelica-Quelltext zum Simulationsergebnis am Beispiel von *OpenModelica* 24

3.1 Synchronisierbare Zeitgitter mit verschiedenen Schrittweiten 29
3.2 Hybride Zeitgitter ohne Rückkopplung (Rückwärts-Synchronisation) . 31
3.3 Hybride Zeitgitter ohne Rückkopplung (Vorwärts-Synchronisation) . . 32
3.4 Hybrides Zeitgitter mit Rückkopplung (Zirkelbeziehung) 32
3.5 Klassenhierarchie von Rechenschritten mit skalarem Ergebnis 37
3.6 Klassenhierarchie von Rechenschritten mit vektoriellem Ergebnis . . . 37
3.7 graphisches Symbol einer Gleichung, gerendert in *MathModelica* 3.0 . 38
3.8 graphisches Symbol einer 2D-Tabelle, gerendert in *MathModelica* 3.0 . 44
3.9 Icon einer Gleichung mit Icon-Overlays, gerendert in *MathModelica* 3.0 53

4.1 Modell des Transmissionsleitwertes L_T 56
4.2 Modelle der Lüftungsleitwerte bei Wohngebäuden 57
4.3 Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit Fensterlüftung $L_{V,FL}$ 58
4.4 Modell des Lüftungsleitwertes bei Nichtwohngebäuden mit raumlufttechnischen Anlagen $L_{V,RLT}$ 59
4.5 Modell des infiltrationsbedingten Lüftungsleitwertes bei Nichtwohngebäuden $L_{V,inf}$ 59
4.6 Modelle der Transmissions- und Lüftungswärmeverluste 60

4.7 Modell der Gesamtwärmeverluste Q_L 61
4.8 Modelle der inneren Wärmegewinne Q_I 62
4.9 Modelle der solaren Wärmegewinne Q_S bei Wohngebäuden 63
4.10 Modelle der solaren Wärmegewinne Q_S bei Nichtwohngebäuden 63
4.11 Visualisierung des Gebäudemodells 65
4.12 Temperaturverläufe während eines Einschwingvorganges: blau: Pufferraumtemperatur Θ_u, rot: Außentemperatur Θ_e, grün: Temperatur der konditionierten Zone Θ_i, magenta: Öffnungszustand der Lüftungsklappe 66
4.13 Wärmeströme während eines Einschwingvorganges: blau: Wärmestrom Q_{ue} von außen in den Pufferraum, rot: Wärmestrom Q_{iu} von der konditionierten Zone in den Pufferraum, grün: Wärmestrom Q in die Speicherkapazität des Pufferraumes, magenta: solare Wärmegewinne Φ durch das Glasdach 67
4.14 Eingangswerte zur Ermittlung der Monatsmitteltemperatur 69
4.15 monatliche Mitteltemperatur 70
4.16 Monatsmitteltemperatur 70
4.17 Jahrestemperaturverläufe: blau: Pufferraumtemperatur Θ_u, rot: Außentemperatur Θ_e, grün: Temperatur der konditionierten Zone Θ_i, magenta: Öffnungszustand der Lüftungsklappe 71
4.18 Jahresverlauf der Wärmeströme von außen in den Pufferraum: blau: solare Wärmegewinne, rot: Transmissionswärmestrom 72
4.19 Jahresverlauf des Wärmestroms von der konditionierten Zone in den Pufferraum: blau: Wärmestrom, rot: monatlich berechneter Mittelwert 73
4.20 Transmissionswärmeverluste der konditionierten Zone (Monatsbilanzverfahren): blau: Gesamt-Transmissionswärmeverluste Q_T, rot: Transmissionswärmeverluste zum Pufferraum 74

Tabellenverzeichnis

2.1 Bautechnische Normen in der Energiekennzahl-Ermittlung 7
2.2 Gebäudetechnische Normen in der Energiekennzahl-Ermittlung 8
2.3 Normen mit Bezug zu den Serien ÖNORM B 8110-1/5/6 und ÖNORM H 5055-5059 10

3.1 Wesentliche Grundelemente des entwickelten Frameworks (Prototypen) 36
3.2 Tabelle 16 aus ÖNORM B 8110-6 (2007) 51