TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMARBEIT

Möglichkeiten der Co-Simulation mit dem Building Controls Virtual Test Bed für den Bereich der objektorientierten Modellbildung physikalischer Systeme

Ausgeführt am Institut für

Analysis und Scientific Computing

der Technischen Universität Wien
unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker

durch

Irene Hafner

Bahnhofstraße 9
7432 Oberschützen

Datum

Unterschrift


STEFAN
OMA
MOTZI
MAMI
PAPI
CHEF
DANKE!
OPA
UNIMENSCHEN
KLEEBLATT
CHRISTOPH
ROMI
AND ALL MY FRIENDS
STUFFY
HISE
BERNHARD
MATTHIAS
OPA
HANNES
JESUS

Kurzfassung

In dieser Arbeit werden die Möglichkeiten der Co-Simulation (kurz für engl. Cooperative Simulation) mithilfe des Co-Simulationstools BCVTB (kurz für Building Controls Virtual Test Bed) untersucht. Basierend auf dem Wunsch der Energieoptimierung in Fertigungsbetrieben soll dem Problem der Simulation eines Gebäudemodells sowie mehreren Maschinenmodellen durch Co-Simulation begegnet werden. Durch eine Co-Simulation können verschiedene Modellbildungsansätze und Teile mit unterschiedlichen Anforderungen an die Simulationsmethodik in einer umfassenden Gesamtsimulation zusammengeführt werden.

Als Einführung wird in den ersten Kapiteln die Methode der objektorientierten Modellbildung physikalischer Systeme, die die Grundlage für die meisten im Zuge dieser Arbeit entwickelten Modelle bildet, beschrieben. Anschließend werden verschiedene Arten der Co-Simulation vorgestellt und ein Überblick über die zugrunde liegende Numerik gegeben, um diese Vorgehensweise hinsichtlich Stabilität und Konsistenz zu rechtfertigen.

Im Hinblick auf die Implementierung wird zunächst auf die Synchronisation der einzelnen Simulatoren mit deren individuellen Solveralgorithmen eingegangen, da diese für eine erfolgreiche Co-Simulation unabdingbar ist. Es folgen detaillierte Beschreibungen der Einzelmodelle, die in der Co-Simulation für den Raum und die darin befindlichen Maschinen verwendet werden, darunter die Entwicklung eines Compartmentmodells für die thermische Modellierung eines Raumes in Modelica via Dymola, verschiedene Maschinenmodelle, die in MATLAB, Simscape und Dymola implementiert wurden, sowie eine mithilfe von Simulink realisierte Temperaturregelung.

In Fallstudien werden nach der Validierung des entwickelten Raummodells zunächst Vor- und Nachteile der Aufspaltung in Einzelsysteme für die Co-Simulation einzelner Maschinenmodelle mit einem Raummodell aufgezeigt. Nach der erfolgreichen Kopplung aller beteiligten Simulatoren wird die Zuverlässigkeit bei der Co-Simulation komplexer Einzelmodelle als auch einer hohen Anzahl an beteiligter Software und Instanzen derer erörtert. Zusätzlich zum Compartmentmodell des Raumes, das im Zuge dieser Arbeit entwickelt wurde, werden anhand eines in EnergyPlus erstellten Gebäudemodells die Unterschiede der beiden Modellierungs- und Simulationsarten dargelegt.

Abschließend werden die Möglichkeiten und Grenzen der Co-Simulation mit BCVTB zusammenfassend aufgeführt, ehe im Schlusswort die Erkenntnisse dieser Arbeit und mögliche weiterführende Methoden zusammengefasst werden.

Abstract

This thesis considers the possibilities of cooperative simulation (abbr. co-simulation) with the co-simulation tool BCVTB (short for Building Controls Virtual Test Bed). For the aim of energy optimization in the manufactoring industry a building model as well as several machine models have to be combined. This problem is faced with co-simulation. Co-simulation enables the overall simulation of models requiring different modelling approaches and hugely differing step sizes or even solver algorithms.

The introductory chapters describe the method of object-oriented modelling of physical systems since most of the included models are implemented based on this approach. Subsequently different co-simulation methods are discussed. Additionally, the basics concerning numerics of co-simulation are described to justify this method considering numerical stability and consistency.

The first part of the implementation section deals with the synchronization of all simulators in spite of their individual solver algorithms since accurate synchronization is necessary to even enable cooperative simulation. The following sections contain a detailed description of all partial models of the room and the machines which will be used in the co-simulation. Thus the development of a thermal compartment model for the machine hall in Modelica via Dymola and individual machine models implemented in MATLAB, Simscape and Dymola as well as a Simulink model for temperature control are described.

The beginning of the following case studies deals with the validation of the Dymola room model and the advantages and disadvantages of separating huge models into partial ones for co-simulation. Besides co-simulating individual machine models with the room model, the BCVTB's performance at the co-simulation of many instances of all simulators and complex partial models is evaluated. Additionally the differences of the modelling and simulation methods of the room model developed in this thesis and an EnergyPlus model of the same building are discussed.

A summary of the possibilities and limits of cooperative simulation with the BCVTB followed by general results of this work and possibilities for further studies concludes the thesis.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Motivation

Im Laufe der letzten Jahre gewann neben dem Verlangen nach maximaler Produktivität auch das Umweltbewusstsein und demzufolge das damit verbundene Interesse an minimalem Energieverbrauch sowohl im Privathaushalt als auch in der Industrie mehr und mehr an Bedeutung.
Um der Forderung nach möglichst geringer Belastung der Umwelt und somit der Energieoptimierung nachkommen zu können, ist es notwendig, die thermischen Vorgänge in Fertigungsbetrieben im Vorhinein abschätzen zu können, was zum Beispiel durch eine Simulation geschehen kann, die sowohl das Verhalten einzelner Maschinen, die sich in einer Halle befinden, als auch die Eigenschaften dieses Raumes und dessen Umgebungsbedingungen berücksichtigt. Die Realisierung einer solchen Simulation hat sich das Projekt INFO[1] (Interdisziplinäre Forschung zur Energieoptimierung in Fertigungsbetrieben) zum Ziel gesetzt. Im Zuge dieses Projekts, an dessen Ausführung sieben Institute der Technischen Universität Wien sowie zehn Firmenpartner beteiligt sind, werden über mehrere Phasen zunächst die in der Halle befindlichen Maschinen detailliert modelliert als auch in weiterer Folge die Verbreitung der von diesen abgegeben Wärme in der Simulation mit einem genauen Modell der Halle und den natürlichen sowie kontrollierten Möglichkeiten zur Kühlung und Heizung derselben, also der thermischen Infrastruktur, über ein gesamtes Jahr beobachtet. Das Modell entsteht anhand einer existenten Maschinenhalle, kann jedoch voraussichtlich mit leicht überschaubarem Aufwand nach den Ansprüchen anderer Betriebe modifiziert werden. Das Projekt INFO wird von der FFG (Österreichische Forschungsförderungsgesellschaft) subventioniert und soll Anfang 2013 abgeschlossen werden.
Da eine Simulation, wie sie im Rahmen des INFO-Projekts vorgesehen ist, die Verbindung von umfangreichen Maschinen- und Gebäudemodellen erfordert, übersteigt dies die Möglichkeiten eines einzelnen Simulators. Deshalb will man diesem Problem mit einer Co-

[1] www.projekt-info.org

Simulation begegnen. In einer Co-Simulation werden über ein Tool wie z.B. dem auf Ptolemy basierenden BCVTB (Building Controls Virtual Test Bed) Modelle gekoppelt, die in unterschiedlicher, auf die jeweiligen Schwerpunkte und Bedürfnisse der einzelnen Modelle zugeschnittenen Software implementiert sind. Mit dieser Methode können insbesondere die einzelnen Maschinen der Fertigungshalle als auch der umgebende Raum separat modelliert und daraufhin in einer Co-Simulation zusammengeführt werden.
Die Idee zur Verfassung dieser Arbeit entstand aus dem Bestreben, das Co-Simulationstool BCVTB, das für die Kopplung der Simulatoren im INFO-Projekt verwendet werden soll, ausgiebig zu testen als auch in weiterer Folge den Mitarbeitern des Projekts in beratender Weise zur Verfügung stehen zu können und somit die Handhabung von BCVTB zu erleichtern sowie den Kopplungsvorgang zu beschleunigen.

1.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit sollen die Möglichkeiten der Co-Simulation mit BCVTB im Hinblick auf die thermische Simulation von Fertigungshallen ausgelotet werden. Zu diesem Zweck wird ein Raum mit als Wärmequelle zu interpretierenden Maschinen simuliert, anhand dessen die zu betrachtenden Fragestellungen untersucht werden können. Die Implementierung des Raumes selbst soll hierbei einerseits mit Dymola und zum anderen mit EnergyPlus erfolgen, um das Verhalten der einzelnen Modellierungsansätze und Simulationsmethoden bei diversen Szenarien zu vergleichen. Einige der im Raum befindlichen Maschinen werden als reine Datenmodelle vorhanden sein, die dem Gesamtmodell über MATLAB zugeführt werden, andere sollen im Detail in Dymola oder Simscape erstellt werden. Zur Regulierung der Temperatur in der Halle soll zudem ein Kühl- bzw. Heizungssystem, dessen Implementierung in Simulink erfolgen wird, mit dem Raum gekoppelt werden.
Abb. 1.1 zeigt die vorgesehene Kommunikation für ein in Dymola implementiertes Raummodell mit drei Maschinenmodellen und einer Temperaturregelung über BCVTB.

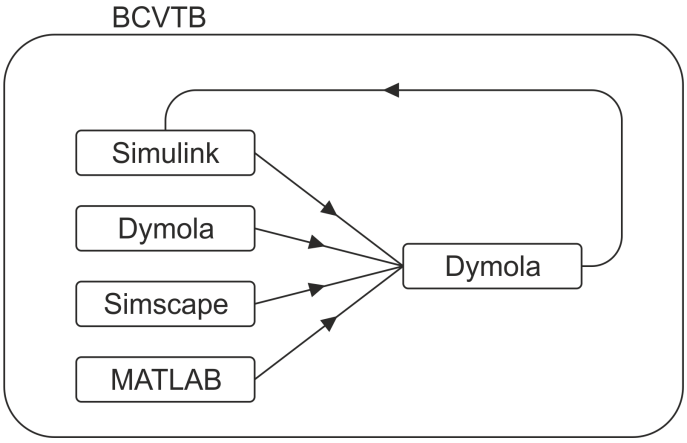

Abbildung 1.1: Vorgesehene Kommunikation der einzelnen Simulatoren für ein Beispielszenario

Die ersten Erkenntnisse sollen dahingehend gewonnen werden, welche Möglichkeiten der Einbindung der verwendeten Simulatoren in BCVTB bestehen. Ein weiteres Hauptaugenmerk liegt auf dem Vergleich der Ergebnisse von Szenarien aus unterschiedlich modellierten Systemen als auch der Simulatoren für die verschiedenen Raummodelle im Bezug auf Rechenzeit und Abweichungen. Abschließend sollen die Grenzen der Co-Simulation mit BCVTB erforscht werden, indem einerseits Maschinen von unterschiedlicher Komplexität eingebunden werden und andererseits die Anzahl an zu koppelnden Modellen variiert wird. Des Weiteren wird der Umgang von BCVTB mit verschiedenen Solverkonfigurationen sowie das Verhalten bei unterschiedlichen Synchronisationseinstellungen untersucht werden.

KAPITEL 2

Physical Modelling

Seit jeher wird versucht, Vorgänge in natürlichen als auch in vom Menschen geschaffenen Systemen durch die Lehre der Physik zu beschreiben. Je komplexer diese Systeme werden, umso interessanter wird die Frage nach dem Verhalten dieser Systeme über einen längeren Zeitraum. Da oftmals nicht die Beobachtung, sondern die Voraussage dieses Verhaltens gewünscht wird, ist es notwendig, diese Systeme zu modellieren, um sie in weiterer Folge in der Hoffnung auf realitätsnahe, verwertbare Resultate simulieren zu können. Ein nahe liegender Ansatz, ein solches Modell zu erstellen, liegt in der Verwendung der für geeignete Systeme bekannten physikalischen Beschreibung derselben: Modellbildung physikalischer Systeme.
Hinsichtlich der Modellbildung physikalischer Systeme muss generell zwischen zwei grundlegend verschiedenen Ansätzen unterschieden werden: kausaler und akausaler Modellbildung. Grundsätzlich geschieht die Berechnung der teilhabenden Variablen in einem kausalen Modell stets durch Zuweisungen, während in einem akausalen Modell die Beziehungen der einzelnen Variablen durch Gleichungen definiert sind, sodass je nach Anforderung in der Simulation jede benötigte Variable aus dem resultierenden Gleichungssystem ausgedrückt und berechnet wird. Genauere Unterschiede dieser beiden Modellbildungsarten werden in diesem Kapitel Schritt für Schritt verdeutlicht werden.

2.1 Kausale Modellbildung

Um die Beschreibung physikalischer Zustände zu erleichtern und eine bessere Übersicht zu erhalten, wird versucht, innerhalb eines kausalen Modells einzelne Zuweisungen zusammenzufassen und diese graphisch in Blöcken mit Ein- und Ausgängen darzustellen. Verbindungen zwischen diesen Blöcken repräsentieren Signalflüsse, deren Richtung so vorgegeben ist, dass sie vom Ausgang eines Blocks zum Eingang eines anderen zeigt; insgesamt kann ein solches Modell somit als Signalflussgraph dargestellt werden. Aus dem Eingangs-

signal eines Blocks wird gemäß den Zuweisungen innerhalb des dem Block zu Grunde liegenden Codes das Ausgangssignal berechnet, welches durch eine Verbindung wiederum einem weiteren Block als Eingangssignal dienen kann. Da die Zustände in jedem Zeitschritt berechnet werden, eignet sich diese Art der Modellierung beispielsweise gut, um Systeme gewöhnlicher Differentialgleichungen darzustellen.
Als Beispiel hierfür kann die Erstellung des Signalflussgraphen für eine Pendelbewegung herangezogen werden. Die Bewegung eines gedämpften Stabpendels mit masselosem Stab wird durch die gewöhnliche Differentialgleichung 2.1 beschrieben.

$$\ddot{\phi} = -\frac{k}{m} \cdot \dot{\phi} - \frac{g}{l} \cdot sin(\phi) \tag{2.1}$$

Hierbei steht l für die Länge des masselosen Stabs, m für die als punktförmig angenommene Masse am Stabende, ϕ für den zwischen der Ruhelage und der aktuellen Position des Pendels eingeschlossenen Winkel, k für die Dämpfungskonstante und g für die Erdbeschleunigung.

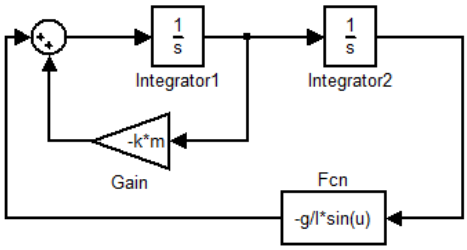

Abbildung 2.1: Signalflussdiagramm der Differentialgleichung für die gedämpfte Pendelbewegung

Das Eingangssignal des ersten Integrator-Blocks entspricht der zweiten Ableitung des Winkels, also der Winkelbeschleunigung. Nach einmaliger Integration, also als Ausgangssignal dieses Blocks, das gleichzeitig das Eingangssignal für den zweiten Integratorblock darstellt, erhält man die Winkelgeschwindigkeit. Der Winkel ϕ ergibt sich aus nochmaliger Integration als Ausgang des zweiten Integratorblocks. Wie Gleichung 2.1 zeigt, erhält man $\ddot{\phi}$ aus der Summe der Winkelgeschwindigkeit mal $-k \cdot m$ mit dem Sinus des Winkels mal $-g/l$. Den ersten Summanden kann man durch Abzweigen des $\dot{\phi}$ entsprechenden Signals und den *Gain*-Block gewinnen; der zweite wird aus dem ϕ entsprechenden Signal im *Fcn*-Block generiert. Die Addition erfolgt in dem runden Block mit zwei Additionszeichen. Somit schließt sich der Kreis, da das so erhaltene Signal der zweiten Ableitung des Winkels entspricht und somit erneut dem Eingang des ersten Integratorblocks dient. Die Zustände jedes dieser Signale werden im Laufe der Simulation zu jedem vom jeweiligen Solver gewählten Zeitschritt berechnet, wofür es selbstverständlich notwendig ist, für die erste Berechnung Anfangswerte des Winkels und der Winkelgeschwindigkeit in den Integratorblöcken anzugeben. Für simpel herzuleitende Differentialgleichungen wie die eben beschriebene kann

dieser Modellbildungsansatz durch gerichtete Graphen somit als durchaus intuitiv und ausreichend erachtet werden.
Um zu veranschaulichen, wie schnell dieser kausale Ansatz zur Modellierung physikalischer Systeme impraktikabel wird, sei als nächstes Beispiel ein einfacher elektrischer Schaltkreis gezeigt. Gegeben sei ein Schaltkreis, der lediglich aus einer Gleichstromquelle und einem Widerstand besteht, wie Abb. 2.2 veranschaulicht.

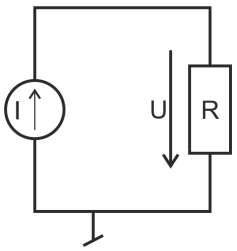

Abbildung 2.2: Schaltbild eines elektrischen Schaltkreises mit einer Gleichstromquelle und einem Widerstand

Da der durch den Widerstand fließende Strom durch die Quelle als konstant vorgegeben ist, kann aus dem Ohmschen Gesetz $U = R \cdot I$, indem diese Gleichung als Zuweisung verstanden wird, die Potentialdifferenz U aus dem Widerstand R und der Stromstärke I berechnet werden. Diese Berechnung kann in einem Signalflussgraphen mit einem einfachen Multiplikationsblock, dessen zwei konstante Eingangssignale die vorgegebene Stromstärke und den Widerstand repräsentieren, realisiert werden, wie Abb. 2.3 zeigt.

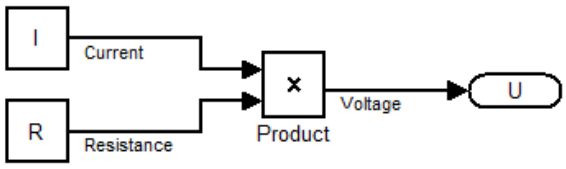

Abbildung 2.3: Signalflussgraph zur Berechnung der Stromstärke aus einem Schaltkreis mit Gleichstromquelle und einem Widerstand

Falls der Schaltkreis insofern abgeändert wird, als man die Gleichstromquelle durch eine Gleichspannungsquelle austauscht, wie Abb. 2.4 veranschaulicht, bewirkt dies in dem zugehörigen Signalflussgraphen eine weit größere Veränderung. Da nun aus der Spannung die Stromstärke berechnet werden soll, muss die Gleichung $U = R \cdot I$ zur Zuweisung $I = \frac{U}{R}$ umgeformt werden. Dies bedeutet für den Signalflussgraphen, dass neben der Abänderung nach dem neuen Spannungseingangssignal auch der Multiplikationsblock gegen einen Divisionsblock getauscht werden muss, wie Abb. 2.5 zeigt.

Sobald man diesem Schaltkreis einen weiteren, parallel geschalteten Widerstand hinzufügt (siehe Abb. 2.6), muss zur Berechnung der Stromstärke die Beziehung $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$ berücksichtigt werden, um mit dem korrekten Gesamtwiderstand zu arbeiten. Der Signal-

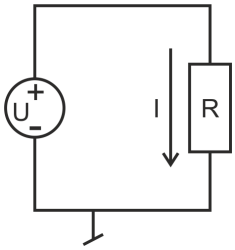

Abbildung 2.4: Schaltbild eines elektrischen Schaltkreises mit einer Gleichspannungsquelle und einem Widerstand

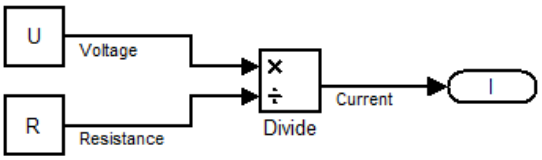

Abbildung 2.5: Signalflussgraph zur Berechnung der Stromstärke aus einem Schaltkreis mit Gleichspannungsquelle und einem Widerstand

flussgraph zu der Berechnung nach diesen Gesetzen ohne weitere Umformung ist in Abb. 2.7 gezeigt.

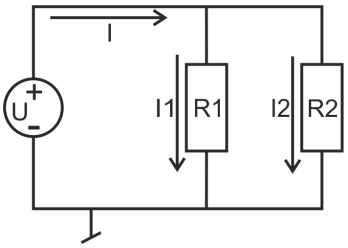

Abbildung 2.6: Schaltbild eines elektrischen Schaltkreises mit Gleichspannungsquelle und zwei parallel geschalteten Widerständen

Neben dem Block, der den Wert des zweiten Widerstandes repräsentiert, wird die Konstante 1 als Eingang für drei zusätzliche Divisionsblöcke benötigt, da zunächst die Kehrwerte von R_1 und R_2 benötigt werden, um aus deren Summe $\frac{1}{R}$ und folglich noch einmal den Kehrwert dieses Resultats zur Bestimmung des Gesamtwiderstandes R zu berechnen. Erst nach dieser umständlichen Vorarbeit kann schließlich die Stromstärke mit dem schon in 2.5 vorhanden gewesenen Divisionsblock mittels $I = \frac{U}{R}$ ermittelt werden.
Es bedarf keiner weiteren Erklärung, um zu erkennen, dass diese Art der Berechnung keineswegs für noch kompliziertere elektrische Kreisläufe oder anspruchsvolle Systeme aus anderen physikalischen Domänen geeignet ist, da bereits für geringfügige Modifikationen

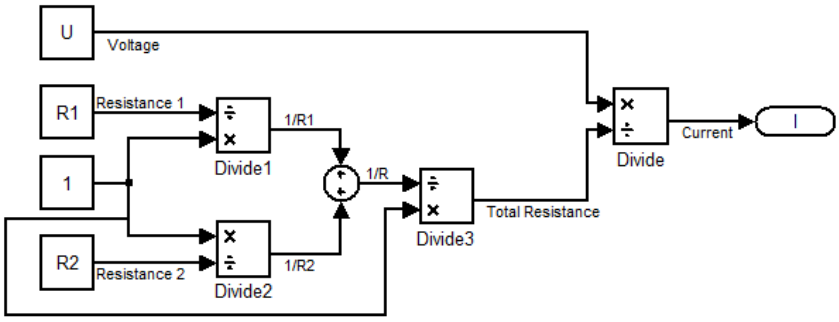

Abbildung 2.7: Signalflussgraph zur Berechnung der Stromstärke aus einem Schaltkreis mit Gleichspannungsquelle und zwei parallel geschalteten Widerständen

im Schaltbild aufwändige Umbauten im Signalflussgraphen oder der Modellierung vorausgehende Gleichungsumformungen notwendig sind. Kapitel 2.2 wird eine äußerst intuitive Modellierungsart für dieses Problem zeigen.

2.2 Akausale Modellbildung

2.2.1 Bondgraphen

Das Konzept der akausalen Modellbildung beruht auf der Theorie der Bondgraphen. Detaillierte Informationen über Bondgraphen können von Kapitel 7 aus [CG91] erhalten werden, aus welchem auch die Information des folgenden kurzen Überblicks entnommen ist. Wie der Name bereits vermuten lässt, können Bondgraphenmodelle ebenfalls graphisch dargestellt werden. Ein Bondgraph besteht per se aus Kanten, genannt Bonds, und Knoten. Die Kanten in einem Bondgraphen haben zunächst keine Kausalität und repräsentieren immer zwei so genannte *konjugierte* Größen, eine *effort*-Variable sowie eine *flow*-Variable, deren Produkt der momentanen Leistung entspricht. Ein Beispiel für solche konjugierten Variablen wären im Fall eines elektrischen Kreislaufs die Spannung bzw. das Potential *U*, welche der *effort*-Variable entspricht, und die Stromstärke *I*, die die konjugierte *flow*-Größe repräsentiert. Deren Produkt $U \cdot I$ ergibt bekannterweise die elektrische Leistung P. In jeder Kante ist eine der beiden Größen durch die andere bestimmt; welche jedoch die abhängige Variable ist, ist vorerst nicht definiert. Je nach den Verbindungen durch Knoten mit anderen Kanten und so genannten Quellen können bei hinreichenden Anfangsbedingungen die *effort*- und *flow*-Variable jeder Kante berechnet werden. Die Gleichungen, die die Berechnung sämtlicher Werte ermöglichen, erhält man aus den Kirchhoffschen Gesetzen: der *Maschengleichung* 2.2 und der *Knotengleichung* 2.3.
Das erste Kirchhoffsche Gesetz besagt, dass die Summe der Flüsse, die zu einem Knoten führen, gleich der Summe der von diesem Knoten abfließenden Flüsse sein muss, was, wenn jeder Fluss mit einem Vorzeichen entsprechend seiner Richtung versehen ist, bedeutet, dass die Gesamtsumme der Flüsse Null ergeben muss.

$$\sum_i I_i = 0 \tag{2.2}$$

Das zweite Kirchhoffsche Gesetz bezieht sich auf *Maschen*. Maschen sind geschlossene Teile eines Graphen, was bedeutet, dass der Anfangspunkt einer Kante beim Durchlaufen in eine fest gewählte Richtung über andere Kanten wieder erreicht werden kann. Das Gesetz besagt, dass die Summe der Spannungen entlang dieser Masche wiederum Null ergeben muss, wodurch beim Erreichen des Ausgangspunkts keine Potentialdifferenz mehr festzustellen ist.

$$\sum_i U_i = 0 \tag{2.3}$$

Diese Gleichungen werden im Fall von Verzweigungen (so genannten *junctions*) im Graphen benötigt, da in diesen per definitionem mehrere Kanten enden oder beginnen. In einem Bondgraphen werden zwei Arten von *junctions* unterschieden: Im Fall einer *0-junction* summieren sich die Flüsse zu Null und die Spannung bleibt gleich (siehe erstes Kirchhoffsches Gesetz), in einer *1-junction* bleibt der Fluss gleich und die Spannungen addieren sich zu Null (entsprechend dem zweiten Kirchhoffschen Gesetz).
Neben diesen Verzweigungen innerhalb des Graphen gibt es Knotenelemente, die mit genau einer Kante verbunden werden, wie Verbraucher- und Speicherelemente sowie Quellen, die entweder die *effort*- oder *flow*-Variable an dem anschließenden Kantenende vorgeben. Verbindungselemente, die mit genau zwei Kanten verbunden werden, sind Gyratoren und Transformatoren, die die Beziehung der Variablen in den beiden anschließenden Kanten durch Gleichungen beschreiben. An dieser Stelle sei darauf hingewiesen, dass tatsächlich eine Gleichung verwendet wird und keine Zuweisung, wie dies in einem gerichteten Signalflussgraphen der Fall wäre. Erst bei der Aufstellung der Gleichungssysteme zur anschließenden Bestimmung aller Variablen in einem Bondgraphen erhalten die an Quellen anliegenden Kanten eine Kausalität (im Beispiel der Verbindung zu einer Spannungsquelle würde somit die Stromstärke zur abhängigen Variablen) und in weiterer Folge wird die Gleichung in einem Knoten so umgeformt, dass die abhängige Variable aus dem Wert der vorgegebenen berechnet werden kann. Insgesamt erhält man durch die Verwendung der eben aufgezählten Gleichungen aller Elemente und Verbindungen ein System, das bei vernünftig gestellten Anfangsbedingungen eindeutig gelöst werden kann.
Abb. 2.8 zeigt den zum Schaltbild mit einer Gleichspannungsquelle und zwei parallel geschalteten Widerständen (siehe Abb.2.6) gehörigen Bondgraphen.

Es ist zu beachten, dass die Halbpfeile am Ende jeder Kante keine Kausalität zu bedeuten haben, sondern lediglich die Stromrichtung vorgeben. Die äußeren Kanten, an denen die Anfangsspannung vorgegeben ist, tragen keine Informationen zum System bei, weshalb man den Bondgraphen aus Abb.2.8 vereinfachen kann, indem man diese Kanten sowie die anschließenden *1-junctions* weglässt. Der vereinfachte Graph mit gleichem Informationsgehalt kann in Abb. 2.9 betrachtet werden.

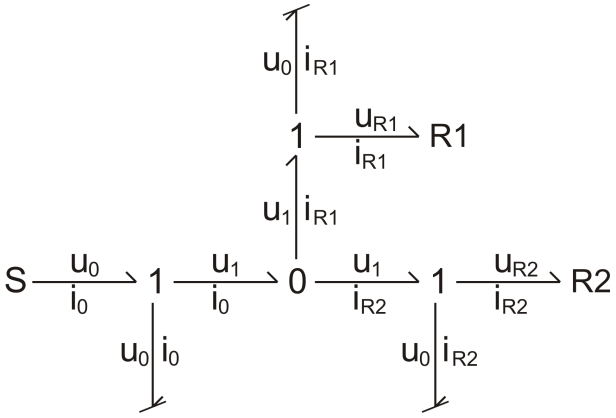

Abbildung 2.8: Bondgraph eines Schaltkreises mit Gleichspannungsquelle und zwei parallel geschalteten Widerständen

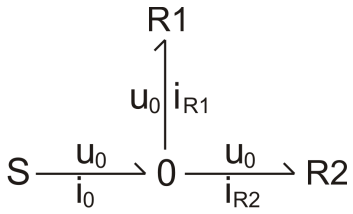

Abbildung 2.9: Vereinfachter Bondgraph eines Schaltkreises mit Gleichspannungsquelle und zwei parallel geschalteten Widerständen

Da von der Vorgabe einer konstanten Anfangsspannung ausgegangen wird, erhält die von der Gleichspannungsquelle ausgehende Kante sogleich eine Kausalität. Die bekannte Größe ist die *effort*-Variable, von der die *flow*-Variable abhängt. Diese Kausalität wird durch einen Strich am selben Ende, an dem sich der Halbpfeil befindet, im Bondgraphen gekennzeichnet. Im Fall einer vorgegebenen *flow*-Variable befände sich die Kennzeichnung an dem Kantenende, das dem Halbpfeil gegenüberliegt. Zur Festlegung der Kausalität der übrigen Kanten wird die *0-junction* betrachtet. Da eine *0-junction* nur eine Gleichung für die Flüsse repräsentiert, kann nur in einer der Kanten der Fluss als abhängige Variable berechnet werden, somit kann sich nur genau ein Kennzeichnungsstrich an diesem Knoten befinden. Bei einer *1-junction* hingegen enden alle Kanten bis auf eine mit einem Strich, da nur in einer die Spannungsgröße berechnet werden kann. Der resultierende, nunmehr kausale Bondgraph ist in Abb. 2.10 dargestellt.

Da u_0 gegeben ist, können aus den beiden mit den Widerständen verbundenen Kanten i_{R1} und i_{R2} mithilfe des Ohmschen Gesetzes berechnet werden, i_0 erhält man aus der Knotengleichung für die *1-junction*. Somit sind letztlich alle Variablen des Systems bekannt.

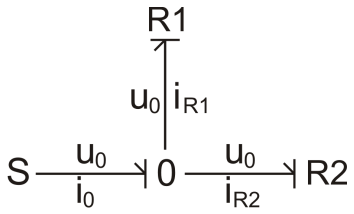

Abbildung 2.10: Bondgraph nach Kausalitätszuweisung der Kanten

2.2.2 Objektorientierte Modellierung physikalischer Systeme

Wie das Beispiel eines elektrischen Schaltkreises aus dem vorhergehenden Kapitel schon vermuten lässt, kann dieser Zugang aus der Bondgraphentheorie auch auf andere physikalische Domänen umgelegt werden. In der Tat kann ein Bondgraph auch domänenübergreifend agieren, was so interpretiert werden kann, dass z.B. elektrische in mechanische Energie umgewandelt wird und fortan mit dieser weitergearbeitet wird.
Die Analoga zu den bereits genannten Größen im elektrischen Schaltkreis für andere Domänen sind in Tabelle 2.1 angeführt, deren Inhalte Kapitel 7 aus [CG91] entnommen sind.

Tabelle 2.1: *effort*- und *flow*-Variablen unterschiedlicher physikalischer Domänen

Domäne	*effort*	*flow*
Elektrizität	Spannung $[V]$	Stromstärke $[A]$
Translationsmechanik	Kraft $[N]$	Geschwindigkeit $[m/s]$
Rotationsmechanik	Drehmoment $[N \cdot m]$	Winkelgeschwindigkeit $[rad/s]$
Hydraulik	Druck $[N/m^2]$	Volumenstrom $[m^3/s]$
Chemie	chemisches Potential $[J/mol]$	molarer Durchfluss $[mol/s]$
Thermodynamik	Temperatur $[K]$	Entropiefluss $[W/K]$

Um wiederverwendbare Modellteile physikalischer Systeme zu erstellen, wird objektorientiert programmiert, was eine akausale Beschreibung der Modelle erfordet. Objektorientierte Programmierung bedeutet, dass jedes Bauteil in einem solchen Modell einer *Instanz* einer gewissen *Klasse* entspricht. Eine Klasse kann zum Beispiel *elektrischer Widerstand* sein, in welcher Eigenschaften des Widerstandes durch Gleichungen und weitere Attribute wie die mögliche Anzahl an Verbindungen über den zugehörigen Source Code festgehalten sind. Um eine Widerstandskomponente in einem Modell zu verwenden, wird ein Repräsentant dieser Klasse erstellt, eine so genannte Instanz, für welche im Folgenden Parameter gewählt und Verbindungen zu anderen Komponenten hergestellt werden können.
Bei objektorientierter Programmierung ist zudem *Vererbung* möglich, was bedeutet, dass durch das *Erben* aus einer Klasse deren Eigenschaften übernommen werden und zudem erweitert werden können. Dies kann beispielsweise für Komponenten, die dieselben Konnektoren besitzen, verwendet werden. Demnach können etwa eine Widerstands- und Spulenklasse von einer Klasse erben, die noch keinen Gleichungsabschnitt besitzt, sodass in den Unterklassen nur noch die definierende Eigenschaft wie $U = R \cdot I$ für den Widerstand

und $U = L \cdot \frac{dI}{dT}$ für die Spule angegeben werden muss.
Zur Verdeutlichung der Signifikanz der Verwendung von gleichungsbasierten Blöcken sei zunächst in Abb. 2.11 das akausale Blockdiagramm zum Schaltbild aus Abb. 2.4 gezeigt.

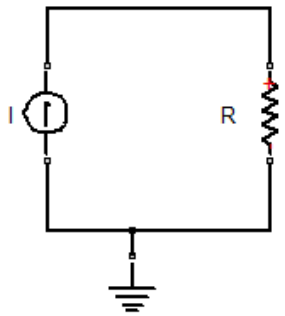

Abbildung 2.11: Akausales Modell eines Schaltkreises mit einer Gleichstromquelle und einem Widerstand

Als erster Vorteil gegenüber dem kausalen Modell erweist sich unmittelbar die Ähnlichkeit des akausalen Modells zu dem zugehörigen Schaltbild. Im Unterschied zu einem Signalflussgraphen ist keine Richtung der Verbindungen zu erkennen, wodurch zu Beginn nicht klar ist, ob in der Widerstandskomponente die aus der Spannung resultierende Stromstärke oder die Spannung aus der Stromstärke berechnet wird, es muss lediglich in jedem Fall die Gleichung $U = R \cdot I$ verwendet werden. Da in unserem Modell eine Gleichstromquelle verwendet wird, folgt in der Simulation die Berechnung der Spannung aus der Stromstärke, indem die Gleichung automatisch passend umgeformt wird. Um die folgende Kausalität in diesem Modell zu ändern, müsste einzig die Komponente der Quelle gegen eine Gleichspannungsquelle ausgetauscht werden (siehe Abb.2.12), sodass ohne weiteres Zutun des Modellerstellers die Gleichung in der Simulation nach der Stromstärke umgeformt wird.

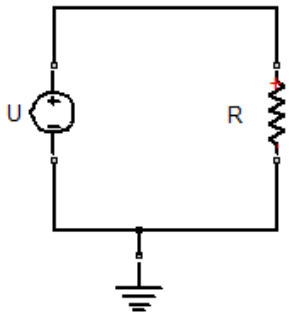

Abbildung 2.12: Akausales graphisches Modell eines Schaltkreises mit einer Gleichspannungsquelle und einem Widerstand

Erweitert man dieses Modell entsprechend dem Schaltbild aus Abb. 2.6 um einen Widerstand in Parallelschaltung, so wird auch im zugehörigen akausalen Modell eine weitere Widerstandsinstanz, die der ersten exakt gleicht, parallel hinzugefügt, wie Abb. 2.13 zeigt.

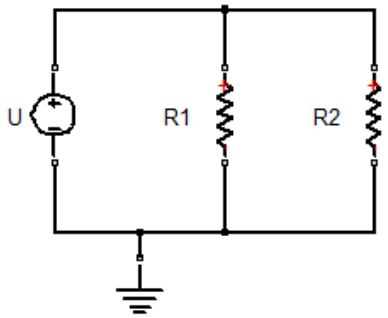

Abbildung 2.13: Akausales Modell eines Schaltkreises mit Gleichspannungsquelle und zwei parallel geschalteten Widerständen

Dies verdeutlicht ein weiteres Mal, dass der Zugang zu physikalischen Systemen über akausale Modellbildung weit intuitiver als auch übersichtlicher gegenüber seinem kausalen Gegenstück ist. Komponenten können einfach wiederverwendet und exakt nach dem Vorbild des korrespondierenden Schaltbildes oder einem Äquivalent einer anderen physikalischen Domäne eingefügt sowie miteinander verbunden werden, ohne dass beim Erstellen des Modells darauf geachtet werden muss, welche Größen bekannt und welche abhängig sind, sodass das Modell auch sofort für andere Fragestellungen mit minimalen Modifikationen weiterverwendet werden kann. Viele Simulatoren, die diese Modellbildungsart verwenden, ordnen zudem allen vorkommenden Größen physikalische Einheiten zu, was den Vorteil hat, dass eine etwaige physikalisch inkonsistente Verwendung von Komponenten beim Kompilationsvorgang sofort zu einer Fehlermeldung führen würde.

2.3 Simulation physikalischer Modelle - Lösung differential-algebraischer Gleichungssysteme

Dies führt sogleich zum nächsten Schritt nach der Erstellung akausaler Modelle: die Simulation derselben. Da es sich bei den auftretenden Gleichungen nicht um reine Differentialgleichungssysteme handelt, können zur Lösung auch nicht ohne Weiteres einfach Differentialgleichungssolver herangezogen werden. Diese Systeme, die sowohl aus rein algebraischen Gleichungen (wie im Beispiel eines elektrischen Schaltkreises für einen Widerstand $U = R \cdot I$) als auch aus gewöhnlichen Differentialgleichungen (z.B. $I = C \cdot \frac{dU}{dt}$ im Fall eines Kondensators) bestehen, werden *differential-algebraische Gleichungssysteme* (engl. *differential algebraic equations*, kurz *DAE*) genannt. Nach dem Aufstellen eines solchen DAE-Systems wird festgestellt, welche Variablen ursprünglich die Unbekannten sind. Falls Variablen nur in differenzierter Form auftreten, wird dieses Differential anstatt der Variablen

als Unbekannte gewählt, wodurch im Anschluss an die Lösung der algebraischen Gleichung ein reines System gewöhnlicher Differentialgleichungen (engl. *ordinary differential equations*, kurz *ODE*) übrig bleibt, welches von ODE-Solvern weiterbehandelt wird.
Das Prinzip von der Aufstellung des DAE-Systems aus einem akausalen Modell und dessen Umformung in Anlehnung an den *Tarjan*-Algorithmus in ein System von Zuweisungen, aus dem alle Unbekannten berechnet werden können, wird anhand eines aus [CK06], Kapitel 7 entnommenen Beispiels erklärt. Gegeben sei das Modell eines Schaltkreises durch Abb. 2.14.

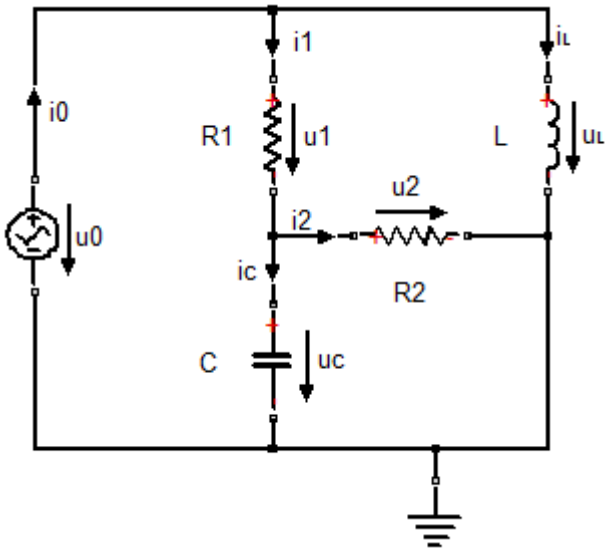

Abbildung 2.14: Modell eines Schaltkreises mit Wechselspannungsquelle, zwei Widerständen, einer Spule und einem Kondensator

Aus diesem Modell können durch die Kirchhoffschen Gesetze (siehe Gleichung 2.2 und 2.3) sowie durch die Gleichungen für Widerstand, Spule und Kondensator die Gleichungen 2.4 bis 2.13 erhalten werden:

$$u_0 = f(t) \tag{2.4}$$
$$u_1 = R_1 \cdot i_1 \tag{2.5}$$
$$u_2 = R_2 \cdot i_2 \tag{2.6}$$
$$u_L = L \cdot \frac{di_L}{dt} \tag{2.7}$$
$$i_C = C \cdot \frac{du_C}{dt} \tag{2.8}$$
$$u_0 = u_1 + u_C \tag{2.9}$$
$$u_L = u_1 + u_2 \tag{2.10}$$

$$u_C = u_2 \tag{2.11}$$
$$i_0 = i_1 + i_L \tag{2.12}$$
$$i_1 = i_2 + i_C \tag{2.13}$$

Anstatt der unbekannten Variablen i_L und u_C werden, wie bereits beschrieben, $\frac{di_L}{dt}$ und $\frac{du_C}{dt}$ als Unbekannte verwendet. Als nächstes werden alle Gleichungen in einer Spalte gegenüber allen Unbekannten in einer weiteren Spalte aufgelistet. Nun wird von jeder Variablen zu jeder Gleichung, in welcher sie vorkommt, eine Linie gezogen. Dadurch erhält man die unter Abb.2.15 zu sehende Struktur.

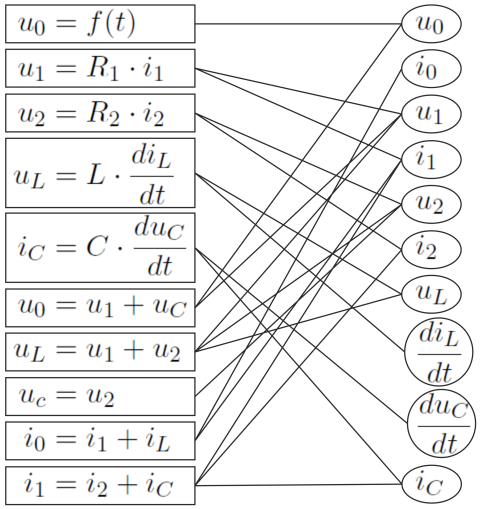

Abbildung 2.15: Strukturdiagramm der Gleichungen und Unbekannten aus Modell 2.14

Bisher handelt es sich bei den angegebenen Gleichungen um akausale Vorschriften, keine Zuweisungen. Da letztendlich jedoch alle Variablen berechnet werden sollen, müssen aus diesen Gleichungen durch Umformungen Zuweisungen erhalten werden. Die Bestimmungen, nach welcher Variablen eine Gleichung umgeformt werden muss, beruhen auf zwei Regeln:
1) Falls in einer Gleichung nur eine Unbekannte vorkommt, wird nach dieser umgeformt. Da nur diese Unbekannte durch diese Gleichung bestimmt werden kann, ist sie entweder durch diese Gleichung festgelegt oder wird gar nicht benötigt.
2) Falls eine Unbekannte nur in einer Gleichung vorkommt, wird diese Gleichung unbedingt zur Bestimmung dieser benötigt, also muss sie nach dieser umgeformt werden, damit der Variablen ein Wert zugewiesen werden kann.
Diese Regeln werden im Tarjan-Algorithmus so umgesetzt, dass zunächst alle Gleichungen gesucht werden, von welchen nur eine schwarze Linie zu wegführt. Wird eine solche Gleichung gefunden, wird diese mit der kleinsten noch verfügbaren Zahl, beginnend bei I, neu nummeriert. Die von der Gleichung wegführende Linie wird daraufhin rot eingefärbt, wäh-

rend alle übrigen Linien, die von der Variablen, in der diese nun rote Linie endet, wegführen, blau eingefärbt werden.
Da in unserem Beispiel die Spannung u_0 von einer gegebenen Funktion bestimmt ist, ist sie die einzige mit 2.4 verbundene Unbekannte. Auch von Gleichung 2.11 führt nur eine schwarze Linie weg, somit werden im ersten Schritt die von 2.4 und 2.11 wegführenden Linien rot und die (in diesem Fall einzige) weitere von u_0 wegführende Linie als auch alle weiteren von u_2 wegführenden Linien blau eingefärbt, wie Abb. 2.16 zeigt. Die Gleichung $u_0 = f(t)$ erhält die neue Nummerierung I, $u_c = u_2$ erhält II als neue Nummer.

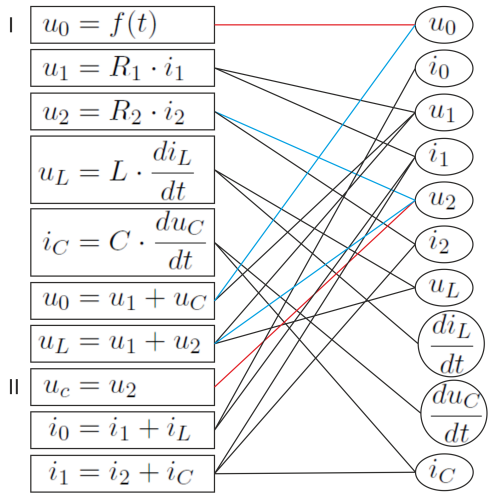

Abbildung 2.16: Strukturdiagramm der Gleichungen und Unbekannten nach der ersten Verwirklichung von Regel Nr.1

Die zweite Regel wird so verwirklicht, dass jede schwarze Linie, die als einzige von einer Variablen wegführt, ebenfalls rot eingefärbt wird, alle von der Gleichung, zu der sie führt, noch wegführenden Linien blau eigefärbt werden und diese Gleichung die höchste noch verfügbare Nummer erhält. In diesem Beispiel ist das für die Unbekannten $i_0, \frac{di_L}{dt}$ und $\frac{du_C}{dt}$ der Fall. Das daraus resultierende Diagramm ist in Abb. 2.17 gezeigt.

Da durch das Einfärben einiger Linien nun wieder Gleichungen und Unbekannte auftreten, von denen nur eine schwarze Linie wegführt, kann der Algorithmus so lange wiederholt werden, bis entweder alle Linien eingefärbt sind oder trotz übriger schwarzer Linien keine von diesen als einzige von einer Gleichung oder Variablen wegführt, was im nächsten Beispiel behandelt werden wird. Im gerade betrachteten Beispiel erhält man zum Schluss lauter eingefärbte Linien, wie in Abb. 2.18 zu sehen ist.

Wird dieses Diagramm jetzt nach der neuen Nummerierung geordnet und jede Gleichung nach der über die rote Linie verbundenen Unbekannten umgeformt, erhält man ein sowohl horizontal als auch vertikal sortiertes Gleichungssystem (siehe Gleichung 2.14 bis 2.23), aus dem folglich bei Ersetzen der Gleichheitszeichen durch Zuweisungen und der Ausführung in dieser Reihenfolge alle Unbekannten berechnet werden.

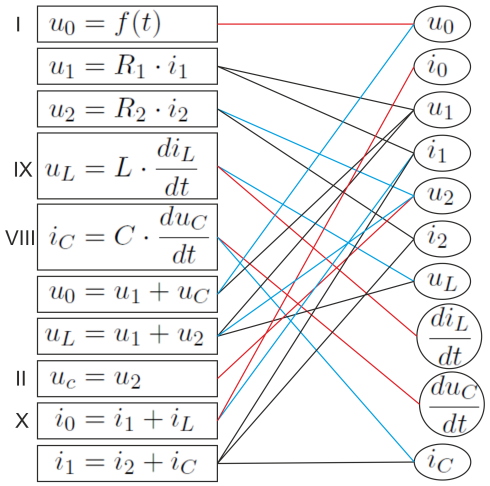

Abbildung 2.17: Strukturdiagramm der Gleichungen und Unbekannten nach der ersten Verwirklichung von Regel Nr.2

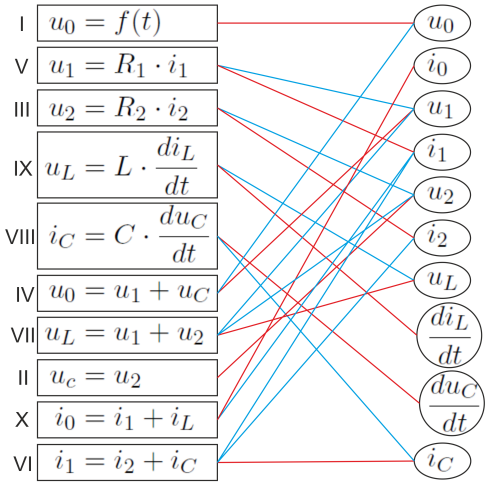

Abbildung 2.18: Strukturdiagramm der Gleichungen und Unbekannten nach Wiederholung der Schritte bis zur Einfärbung aller Linien

$$u_0 = f(t) \tag{2.14}$$
$$u_2 = u_C \tag{2.15}$$
$$i_2 = \frac{u_2}{R_2} \tag{2.16}$$
$$u_1 = u_0 - u_C \tag{2.17}$$

$$i_1 = \frac{u_1}{R_1} \tag{2.18}$$
$$i_C = i_1 - i_2 \tag{2.19}$$
$$u_L = u_1 + u_2 \tag{2.20}$$
$$\frac{du_c}{dt} = \frac{i_C}{C} \tag{2.21}$$
$$\frac{di_L}{dt} = \frac{u_L}{L} \tag{2.22}$$
$$i_0 = i_1 + i_L \tag{2.23}$$

Wie bereits kurz erwähnt, kann es passieren, dass dieser Algorithmus vor der Einfärbung aller Linien terminiert und somit keine Berechnung erfolgen kann. Einer der Gründe für einen solchen vorzeitigen Abbruch kann das Auftreten einer *algebraischen Schleife* sein. Eine algebraische Schleife tritt auf, sofern einige Variablen durch Gleichungen so von einander abhängen, dass, egal bei welcher Gleichung begonnen wird, man bei dem Versuch, deren Variablen aus den restlichen Gleichungen zu gewinnen, schlussendlich wieder bei der anfänglichen Gleichung angelangt, sodass diese schließlich zur Lösung von sich selbst benötigt würde. Es tritt also ein Teilsystem von Gleichungen auf, die simultan gelöst werden müssen. Der einfachste Fall, wie eine algebraische Schleife auftreten könnte, wäre somit ein System der Form

$$v_1 = f(v_2)$$
$$v_2 = f(v_1)$$

Abb. 2.19 zeigt einen im Vergleich zu Abb. 2.14 nur minimal abgeänderten Schaltkreis, für dessen Gleichungssystem der Tarjan-Algorithmus eine algebraische Schleife aufzeigen wird. Dieses Beispiel wurde ebenfalls aus [CK06], Kapitel 7, entnommen.

Im resultierenden Gleichungssystem tritt im Unterschied zum vorhergehenden Beispiel nur eine weitere Widerstandsgleichung ($u_3 = R_3 \cdot i_3$) anstatt der Gleichung für den Kondensator (2.8) auf. Die Anwendung des Tarjan-Algorithmus auf dieses System führt zu der in 2.20 dargestellten Struktur, bei der von jeder übrigen Gleichung als auch Variablen mindestens zwei schwarze Linien wegführen.

Um das System aus den verbleibenden sechs Gleichungen dennoch zu lösen, kann der *Tearing*-Algorithmus verwendet werden.

$$u_1 = R_1 \cdot i_1 \tag{2.24}$$
$$u_2 = R_2 \cdot i_2 \tag{2.25}$$
$$u_3 = R_3 \cdot i_3 \tag{2.26}$$
$$u_0 = u_1 + u_3 \tag{2.27}$$
$$u_3 = u_2 \tag{2.28}$$
$$i_1 = i_2 + i_3 \tag{2.29}$$

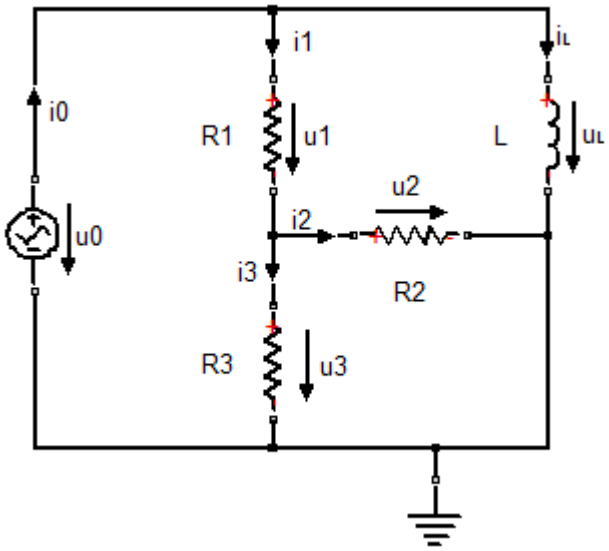

Abbildung 2.19: Modell eines Schaltkreises mit Wechselspannungsquelle, drei Widerständen und einer Spule

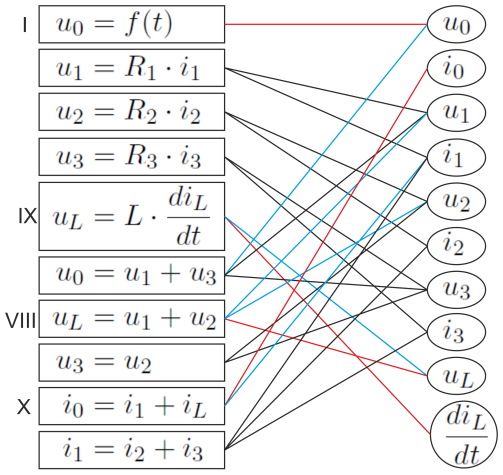

Abbildung 2.20: Strukturdiagramm für Modell 2.19 nach Anwendung des Tarjan-Algorithmus

Bei diesem Algorithmus wird so vorgegangen, dass zunächst eine der verbleibenden Unbekannten als *Tearing-Variable* gewählt wird, in diesem Beispiel wird i_3 ausgesucht. Wenn man annimmt, diese Variable zu kennen, können auch alle anderen Unbekannten ermittelt werden. Bei Systemen, in denen keine Variable gewählt werden kann, durch die alle restlichen bestimmt werden könnten, wird versucht, eine Variable zu wählen, durch die möglichst

viele andere ermittelt werden könnten; für die verbleibenden muss diese Vorgehensweise wiederholt werden. Methoden zur Wahl der ersten Tearing-Variable, die ermöglicht, dass insgesamt nur wenige Tearing-Variablen benötigt werden, werden in dieser Arbeit nicht weiter erörtert, weiterführende Informationen können [CK06] entnommen werden. Neben der Tearing-Variable wird zudem eine Gleichung ausgewählt, in der diese Variable auftritt. In unserem Beispiel wird hierfür 2.29 gewählt. Da die Variable i_3 noch nicht bekannt ist, muss sie iteriert werden. Hierzu ersetzt man in Gleichung 2.29 i_3 durch $i_{3_{neu}}$, belässt es jedoch in allen anderen Gleichungen, in welchen i_3 vorkommt. Im neuen Gleichungssystem 2.30 bis 2.35 steht i_3 für eine erste Schätzung und $i_{3_{neu}}$ für die nächste Verbesserung dieser.

$$u_1 = R_1 \cdot i_1 \tag{2.30}$$
$$u_2 = R_2 \cdot i_2 \tag{2.31}$$
$$u_3 = R_3 \cdot i_3 \tag{2.32}$$
$$u_0 = u_1 + u_3 \tag{2.33}$$
$$u_3 = u_2 \tag{2.34}$$
$$i_{3_{neu}} = i_1 - i_2 \tag{2.35}$$

Zur Ermittlung von i_3 werden nun die Nullstellen von $F = i_{3_{neu}} - i_3$ mithilfe des Newton-Verfahrens iteriert. Die Hesse-Matrix (die wie die Funktion F in diesem Fall ein Skalar ist) dieser Funktion kann durch algebraisches Differenzieren des gesamten Gleichungssystems gewonnen werden, mit deren Hilfe daraufhin die nächste Iteration für i_3 aus

$$i_3 = i_3 - \frac{H}{F}$$

berechnet werden kann.
Vor der Erörterung weiterer möglicher Probleme bei der Lösung differential-algebraischer Gleichungssysteme seien noch die Definitionen einiger Begriffe eingeführt. Ein erster Wert, der salopp gesagt für die Entfernung eines differential-algebraischen Gleichungssystems zu einem System gewöhnlicher Differentialgleichungen steht, ist der *differentielle Index*. Laut [Fri04], Kapitel 18, ist der differentielle Index wie folgt definiert:
Der differentielle Index eines DAE-Systems gibt an, wie oft gewisse Gleichungen aus dem DAE-System mindestens differenziert werden müssen, um ein reines ODE-System zu erhalten.
Ein gewöhnliches Differentialgleichungssystem der Form

$$\dot{x}(t) = f(t, x(t))$$

hat demnach Index 0.
Ein System der Form

$$\dot{x}(t) = f(t, x(t), y(t)) \tag{2.36}$$
$$0 = g(t, x(t), y(t)) \tag{2.37}$$

hat Index 1, sofern die Jacobi-Matrix $\partial g/\partial y$ regulär ist, da in diesem Fall die zweite Gleichung (2.37) differenziert und nach $\dot{y}$ umgeformt werden kann, siehe Gleichung 2.38:

$$\dot{y} = -\left(\frac{\partial g}{\partial y}\right)^{-1}\left(\frac{\partial g}{\partial x}\right)\dot{x} - \left(\frac{\partial g}{\partial y}\right)^{-1}\left(\frac{\partial g}{\partial t}\right) \tag{2.38}$$

Durch Einsetzen für $\dot{y}$ in Gleichung 2.36 erhält man eine gewöhnliche Differentialgleichung, die durch einen passenden ODE-Solver gelöst werden kann.
Für Probleme von höherem Index, wie sie im Bereich des Physical Modelling oft auftreten, wird eine Index-Reduktion durchgeführt, bis das Problem auf eines von Index 1 reduziert wurde, wonach wieder nach der obigen Methode vorgegangen werden kann.
Als Beispiel für ein Problem von höherem Index, das aus Kapitel 7 von [CK06] entnommen wurde, sei der in Abb. 2.21 dargestellte Schaltkreis gegeben.

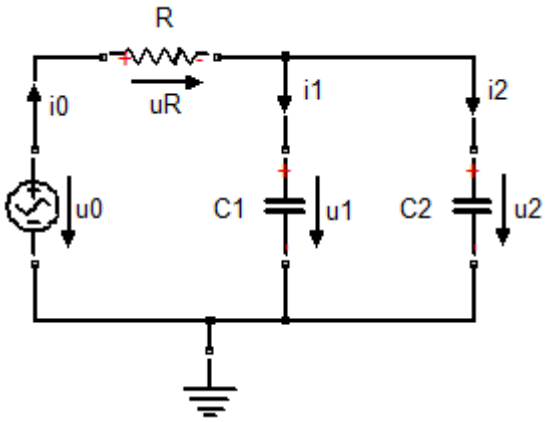

Abbildung 2.21: Modell eines Schaltkreises mit Wechselspannungsquelle, einem Widerstand und zwei Kondensatoren

Als Gleichungssystem zu dessen Beschreibung erhält man Gleichung 2.39 bis 2.45.

$$u_0 = f(t) \tag{2.39}$$
$$u_R = R \cdot i_0 \tag{2.40}$$
$$i_1 = C_1 \cdot \frac{du_1}{dt} \tag{2.41}$$
$$i_2 = C_2 \cdot \frac{du_2}{dt} \tag{2.42}$$
$$u_0 = u_R + u_1 \tag{2.43}$$
$$u_2 = u_1 \tag{2.44}$$
$$i_0 = i_1 + i_2 \tag{2.45}$$

Nach Anwendung des Tarjan-Algorithmus erhält man zwei blaue Linien, die zu i_2 führen, jedoch keine rote (siehe Abb.2.22), was bedeutet, dass keine Gleichung bleibt, um i_2 zu berechnen; es tritt eine so genannte strukturelle Singularität (engl. *structural singularity*) auf.

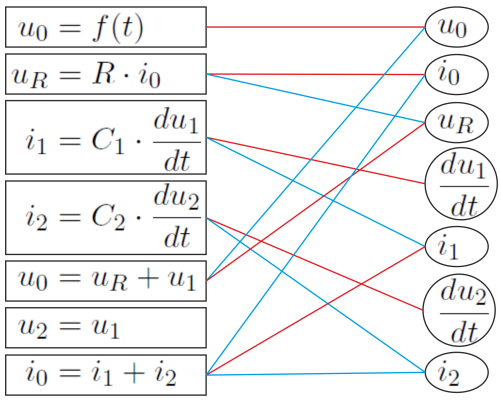

Abbildung 2.22: Strukturdiagramm für das Resultat des Tarjan-Algorithmus angewandt auf Gleichung 2.39 bis 2.45

Weiters bleibt Gleichung 2.44 unbenutzt. Diese Gleichung repräsentiert eine Zwangsbedingung und wird im Englischen *constraint equation* genannt. Der Idee von Costas Pantelides folgend könnte diese Zwangsbedingung einfach differenziert werden, immerhin muss wegen der Gleichheit von u_1 und u_2 zu jedem Zeitpunkt auch $\frac{du_1}{dt} = \frac{du_2}{dt}$ gelten. Durch die Abänderung dieser Gleichung bliebe sie nicht unverwendet und alle Unbekannten würden wieder einer Gleichung zugeordnet; das System hätte Index 1 und wäre somit mit bekannten Methoden lösbar. Allerdings geht durch die Differentiation die Anfangsbedingung für u_1 und u_2 verloren, was offenkundig wiederum nicht wünschenswert ist.
Eine andere Methode zur Behandlung solcher Probleme stellt die Verwendung von Pseudovariablen, so genannten *dummy derivatives*, dar. Bei dieser Vorgehensweise wird zwar die Zwangsbedingung differenziert, jedoch werden sowohl die ursprüngliche als auch die differenzierte Gleichung behalten. Um dennoch kein überbestimmtes System zu erhalten, wird des Weiteren eine der nun differenzierten Variablen, in unserem Fall wird o.B.d.A. du_2/dt gewählt, durch eine zusätzliche Variable ersetzt, die in diesem Beispiel mit du_2 bezeichnet wird. Als neues Gleichungssystem erhält man somit Gleichung 2.46 bis 2.53:

$$u_0 = f(t) \tag{2.46}$$
$$u_R = R \cdot i_0 \tag{2.47}$$
$$i_1 = C_1 \cdot \frac{du_1}{dt} \tag{2.48}$$
$$i_2 = C_2 \cdot du_2 \tag{2.49}$$
$$u_0 = u_R + u_1 \tag{2.50}$$

$$u_2 = u_1 \tag{2.51}$$

$$du_2 = \frac{du_1}{dt} \tag{2.52}$$

$$i_0 = i_1 + i_2 \tag{2.53}$$

Dieses DAE-System hat den differentiellen Index 1 (was bedeutet, dass das ursprüngliche System vom Index 2 war) und kann somit mit den bereits vorgestellten Methoden gelöst werden.

KAPITEL 3

Co-Simulation und Multirate-Simulation

Da die Modellierung umfangreicher Systeme mit Teilmodellen unterschiedlicher Komplexität, die zudem verschiedene Modellbildungsansätze erfordern, die Möglichkeiten eines einzelnen Simulators oft überschreitet, ist es in diesen Fällen ratsam, einzelne Teilmodelle in unterschiedlicher Software zu implementieren und diese daraufhin gekoppelt zu simulieren. Sobald an einer Simulation zwei oder mehr verschiedene Simulatoren beteiligt sind, wird von einer Co-Simulation (Abk. für engl. *Cooperative Simulation*) gesprochen. Für gewöhnlich steht über einer Co-Simulation eine Software, die die Kommunikation aller verwendeten Simulatoren steuert. Sofern die einzelnen Simulatoren mit individuellen Schrittweiten arbeiten, die im Allgemeinen nur zu den Synchronisationszeitpunkten übereinstimmen, spricht man zudem von einer Multirate-Simulation.
Multirate-Simulation ist auch innerhalb eines Simulators möglich, wenn für Teile des Modells ein eigener Solver benutzt wird, wie es z.B. in der MATLAB-Toolbox Simscape möglich ist. Auf diese Weise kann den Ansprüchen einzelner Teile eines Modells, die zwar die gleiche Modellierungsart, jedoch etwa aus Steifigkeitsgründen einen anderen Solver oder zumindest einen anderen Zeitschritt als das restliche Modell benötigen, entsprochen werden. Für Multirate-Simulation ist es demnach nicht erforderlich, mehrere Simulatoren zu koppeln, jedoch sind mehrere Solver mit unterschiedlichen Zeitschritten an der Simulation beteiligt. Bei einer Kopplung von verschiedenen Simulatoren, die individuelle Zeitschritte setzen, überschneiden sich die Begriffe der Co- und Multirate-Simulation.

3.1 Numerische Grundlagen der Co-Simulation

Laut [Völ10] unterscheidet man in der Multirate-Simulation grundlegend zwischen *loose coupling* und *strong coupling*. Beim *loose coupling* erfolgt zu fixen Zeitpunkten ein Daten-

austausch zwischen den Teilsystemen, wobei die benötigten Werte aus anderen Simulatoren aus vorhergehenden Werten extrapoliert werden müssen. Beim *strong coupling* erfolgt bei jedem Zeitschritt ein Austausch in jedem Iterationsschritt, bis gewünschte Genauigkeitsvoraussetzungen erfüllt sind. Diese Art der Simulator-Kopplung ist gegenüber der Methode des *loose coupling* zwar genauer, jedoch dementsprechend langsamer und erfordert zudem Eingriffe in die jeweiligen Solveralgorithmen, weshalb sie in dieser Arbeit nicht weiter behandelt wird. Die meisten heutzutage dokumentierten Co-Simulationsmethoden erfolgen über die Verwendung eines Solvers für das Gesamtsystem, der den Datenaustausch zu fest vorgegebenen Zeitschritten in konstanten Abständen veranlasst. Als weitere Möglichkeit kann über der Gesamtsimulation ein Solver stehen, der die Kommunikationszeitschritte je nach Genauigkeitsansprüchen variiert und demnach auch Schritte wieder verwerfen kann, wodurch die teilhabenden Simulatoren gezwungen werden, ab einem gewissen Zeitpunkt die interne Simulation zu wiederholen, um zum vom Gesamtsolver neu berechneten Zeitpunkt zu kommunizieren. Diese Art der Simulator-Kopplung erfordert Möglichkeiten zur Beeinflussung bereits zurückliegender Zeitschritte in den einzelnen Simulatoren durch eine Vorgabe von außen, was nur durch Eingriffe in die Solveralgorithmen realisiert werden kann. Für Interessierte sei an dieser Stelle auf [TS12] verwiesen, wo für Co-Simulation durch Functional Mockup Interfaces (siehe [Assa]) die Fehlerschätzung erörtert wird.
Für das *loose coupling* werden zwei Methoden der Kommunikation unterschieden. Beim *Gauß-Seidl*-Typ wird so vorgegangen, dass die Simulation der Teilsysteme für ein Kommunikationsintervall hintereinander erfolgt. Dieses Schema sei im Folgenden anhand eines Beispiels für zwei Teilsysteme, welches [Völ10] entnommen wurde, erörtert. Angenommen, System 1 sei durch die Gleichungen 3.1 - 3.2 und System 2 durch 3.3 - 3.4 für die Zustandsvektoren x_1, y_1, x_2 und y_2 beschrieben.

$$\dot{x_1} = f_1(x_1, y_2) \tag{3.1}$$
$$y_1 = g_1(x_1, y_2) \tag{3.2}$$
$$\dot{x_2} = f_2(x_2, y_1) \tag{3.3}$$
$$y_2 = g_2(x_2, y_1) \tag{3.4}$$

Die Elemente von $y_i, \quad i = \{1, 2\}$ werden vom jeweils anderen System zur Berechnung der Zustände im eigenen System benötigt. Die Synchronisation soll zu äquidistanten Zeitpunkten $\{t_0, t_1, \ldots, t_{n-1}, t_n\}$ geschehen. Bei der ersten Kommunikation werden die gegebenen Anfangswerte beider Teilsysteme ausgetauscht. Für jede weitere Synchronisation wird wie folgt vorgegangen: Zunächst wird eines der Systeme, in unserem Fall o.B.d.A System 1 gewählt. Dieses System wird vom aktuellen Zeitpunkt t_j bis zum nächsten Synchronisationszeitpunkt t_{j+1} entsprechend seinem Solver numerisch integriert, wobei der noch unbekannte Wert von y_2 in diesem Intervall aus den bereits bekannten Werten zu vorhergehenden Zeitpunkten extrapoliert wird. Nach dem Erreichen von t_{j+1} wird dieser an das zweite System übergeben, welches nun den Wert von y_1 für das Synchronisationsintervall $[t_j, t_{j+1}]$ interpolieren und danach alle Zustände entsprechend seines Solvers berechnen kann. Diese Vorgehensweise ist in Abb.3.1 dargestellt.

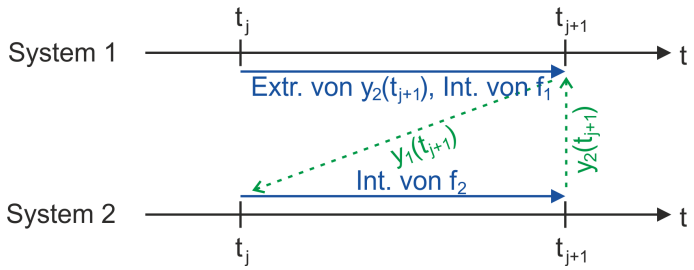

Abbildung 3.1: Darstellung der Vorgangsweise des Gauß-Seidl-Typs des *loose coupling*

Die Wahl des Systems, das zuerst ausgeführt wird, kann nach der Steifigkeit der Systeme entschieden werden. Bei der *fastest first*-Methode wird zuerst das steifere System integriert und somit werden die langsam ansteigenden Größen zuerst extrapoliert, was einen geringeren Fehler liefert, aufgrund der kleineren Kommunikationsschrittweite ist damit jedoch ein hoher Speicheraufwand verbunden. Die umgekehrte Vorgehensweise wäre *slowest first*, bei der die sich schnell ändernden Größen extrapoliert werden, was den Fehler erhöht, den Speicheraufwand jedoch reduziert, da die Kommunikationsschrittweite größer gewählt werden kann.
Bei der zweiten Möglichkeit des *loose coupling*, dem Jacobi-Typ, erfolgt die Simulation aller Systeme zeitgleich; die benötigten Zustände aus anderen Systemen müssen demnach für jede Teilsimulation extrapoliert werden. Das Schema dieser Methode zeigt Abb.3.2.

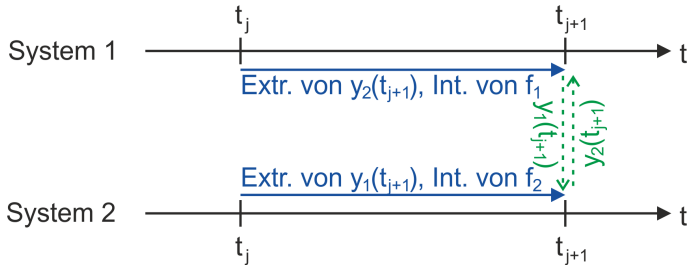

Abbildung 3.2: Darstellung der Vorgangsweise des Jacobi-Typs des *loose coupling*

Auf die Konsistenz und Stabilität bei der Methode des *loose coupling* sei nun näher eingegangen, wobei sämtliche Informationen hierzu aus [Tr8] und [Mel08] entnommen sind. Es sei ein Anfangswertproblem der Form

$$\dot{y}(t) = f(t, y(t)) \tag{3.5}$$

$$y(t_0) = y_0, \tag{3.6}$$

wobei y einer vektorwertigen Funktion entspricht, gegeben. Im Falle der Lipschitzstetigkeit von f existiert auf einem vorgegebenen Intervall $[t_0, t_n]$ nach Picard-Lindelöf eine eindeutige Lösung. Die Approximation y_i der Lösung $y(t_i), \quad i = 1, \ldots n$ zu jedem Zeitschritt wird bei

der Verwendung eines Mehrschrittverfahrens aus folgender Formel gewonnen:

$$\sum_{j=0}^{k} \alpha_{k-j} y_{i+1-j} = \Delta t \cdot \Phi_f(t_{i+1-j}, y_{i+1-j}, \Delta t), \tag{3.7}$$

wobei Φ die Inkrementfunktion des Verfahrens und Δt den in diesem Fall äquidistant gewählten Abstand zwischen t_i und t_{i-1} für $i = 1, \ldots n$ bezeichnet.
Der Begriff der Konsistenz misst den Fehler, den ein Verfahren in einem Schritt macht. Der Konsistenzfehler τ ist definiert durch

$$\tau^{i+k}(\Delta t) := \sum_{j=0}^{k} \alpha_{k-j} y(t_{i+1-j}) - \Delta t \cdot \Phi_f(t_{i+1-j}, y(t_{i+1-j}), \Delta t). \tag{3.8}$$

Ein Verfahren heißt konsistent, falls für jede Wahl der Anfangswerte

$$\lim_{\Delta t \to 0} \left(\frac{\tau^{i+k}(\Delta t)}{\Delta t} \right) = 0 \tag{3.9}$$

erfüllt ist.
Ein Verfahren heißt konsistent der Ordnung p, falls eine Konstante $C > 0$ existiert, sodass

$$\left\| \tau^{i+k}(\Delta t) \right\| \leq C \cdot (\Delta t)^{p+1} \tag{3.10}$$

oder äquivalent dazu

$$\left\| \frac{\tau^{i+k}(\Delta t)}{\Delta t} \right\| \leq C \cdot (\Delta t)^{p} \tag{3.11}$$

gilt.
Ein weiterer Begriff, der ein numerisches Verfahren charakterisiert, ist die Nullstabilität. Ein nullstabiles Verfahren liefert für die Gleichung $\dot{y}(t) = 0$ bei beliebiger Wahl der Anfangswerte eine beschränkte Funktion. Zur Bestimmung der Nullstabilität wird das erste charakteristische Polynom $\rho(\zeta)$ des Verfahrens benötigt:

$$\rho(\zeta) := \sum_{j=0}^{k} \alpha_j \zeta^j \tag{3.12}$$

Ein Verfahren heißt nullstabil, falls für jede Nullstelle λ des ersten charakteristischen Polynoms $|\lambda| \leq 1$ gilt und jede Nullstelle mit $|\lambda| = 1$ einfach ist.
Zur Untersuchung der Konsistenz und Nullstabilität eines *loose coupling* -Co-Simulationsverfahrens sei ein lineares Einschrittverfahren betrachtet, für welches die aus Gleichung 3.7 gewonnene Gleichung des Verfahrens wie folgt aussieht:

$$\alpha_0 y_i + \alpha_1 y_{i+1} = \Delta t \left(\beta_0 f(t_i, y_i) + \beta_1 f(t_{i+1}, y_{i+1}) \right) \tag{3.13}$$

Als Konsistenzfehler erhält man für dieses Verfahren

$$\tau^{i+1}(\Delta t) = \alpha_0 y(t_i) + \alpha_1 y(t_{i+1}) - \Delta t \left(\beta_0 f(t_i, y(t_i)) + \beta_1 f(t_{i+1}, y(t_{i+1})) \right). \tag{3.14}$$

Im Fall einer Co-Simulation, in der einige Einträge von $y(t_{i+1})$ aus einem anderen Teilsystem kommen und demnach noch nicht bekannt sind, muss $y(t_{i+1})$ extrapoliert werden. Der extrapolierte Wert sei mit $y_c(t_{i+t})$ bezeichnet. Damit ergibt sich als Konsistenzfehler $\tau_c^{i+1}(\Delta t)$:

$$\begin{aligned}\tau_c^{i+1}(\Delta t) =& \alpha_0 y(t_i) + \alpha_1 y(t_{i+1}) - \Delta t\,(\beta_0 f(t_i, y(t_i)) + \beta_1 f(t_{i+1}, y_c(t_{i+1})))\\ =& \alpha_0 y(t_i) + \alpha_1 y(t_{i+1}) - \Delta t(\beta_0 f(t_i, y(t_i)) + \beta_1 f(t_{i+1}, y(t_{i+1}))\\ & - \beta_1 f(t_{i+1}, y(t_{i+1})) + \beta_1 f(t_{i+1}, y_c(t_{i+1})))\\ =& \tau^{i+1}(\Delta t) + \Delta t \cdot \beta_1 \cdot (f(t_{i+1}, y(t_{i+1})) - f(t_{i+1}, y_c(t_{i+1})))\end{aligned}$$

Zur Feststellung der Konsistenz sei die Norm von $\tau_c^{i+1}(\Delta t)$ dividiert durch Δt betrachtet:

$$\begin{aligned}\left\|\frac{\tau_c^{i+k}(\Delta t)}{\Delta t}\right\| &= \left\|\frac{\tau^{i+k}(\Delta t)}{\Delta t} + \beta_1 \cdot (f(t_{i+1}, y(t_{i+1})) - f(t_{i+1}, y_c(t_{i+1})))\right\|\\ &\leq \left\|\frac{\tau^{i+k}(\Delta t)}{\Delta t}\right\| + |\beta_1| \cdot \|f(t_{i+1}, y(t_{i+1})) - f(t_{i+1}, y_c(t_{i+1}))\|\\ &\leq \left\|\frac{\tau^{i+k}(\Delta t)}{\Delta t}\right\| + L \cdot |\beta_1| \cdot \|y(t_{i+1}) - y_c(t_{i+1})\|\end{aligned}$$

Hierbei folgt die letzte Ungleichung aus der Lipschitzstetigkeit der Funktion f mit der Lipschitzkonstanten L. Wird für die Approximation $y_c(t_{i+1})$ einfach $y(t_i)$ gewählt, was einer konstanten Fortsetzung entspricht, und betrachtet man die Taylorentwicklung von $y(t_{i+1})$ und $y(t_i)$ um $t_i + \alpha\Delta t$ für beliebiges $\alpha \in (0, 1)$ (Gleichungen 3.15 bzw.3.16), so sieht man, dass deren Differenz Terme der Ordnung 1 und höher von Δt ergibt (vgl. Gleichung 3.17).

$$y(t_{i+1}) = y(t_i + \alpha\Delta t) + (1-\alpha)\Delta t\dot{y}(t_i + \alpha\Delta t) + \frac{((1-\alpha)\Delta t)^2}{2}\ddot{y}(t_i + \alpha\Delta t) + \dots \quad (3.15)$$

$$y(t_i) = y(t_i + \alpha\Delta t) - \alpha\Delta t\dot{y}(t_i + \alpha\Delta t) + \frac{(\alpha\Delta t)^2}{2}\ddot{y}(t_i + \alpha\Delta t) + \dots \quad (3.16)$$

$$\left\|\frac{\tau_c^{i+k}(\Delta t)}{\Delta t}\right\| \leq \left\|\frac{\tau^{i+k}(\Delta t)}{\Delta t}\right\| + L|\beta_1| \cdot O(\Delta t) \quad (3.17)$$

Für die Konsistenzordnung bedeutet dies, dass für ein Verfahren der Ordnung 1 die Ordnung bei einer Co-Simulation erhalten bleibt und somit kein Genauigkeitsverlust zu verzeichnen ist; bei Verfahren höherer Ordnung erhält man zwar auch Konsistenz, jedoch von niedriger Ordnung.
Zur Überprüfung der Nullstabilität sei wieder das Verfahren aus Gleichung 3.13 betrachtet. Das charakteristische Polynom hängt lediglich von der linken Seite dieser Gleichung ab, die Co-Simulation hat jedoch nur Auswirkungen auf die Auswertung der Funktion f bei Verwendung von $y(t_{i+1})$ und somit die rechte Seite dieser Gleichung. Nullstabilität wird somit durch die Aufteilung in eine Co-Simulation nicht beeinflusst, allerdings nur, solange es sich um

reine ODE-Systeme handelt. Für DAE-Systeme ist laut [Tr8] aus [KS00] bekannt, dass bei zwei gekoppelten Systemen Nullstabilität erhalten bleibt, sofern in mindestens einem System dessen Outputs nur von den Zustandsvariablen und keinen algebraischen Funktionen der Inputs abhängen.

3.2 Co-Simulation mit BCVTB

Die in dieser Arbeit verwendete Art der Co-Simulation beschränkt sich im Wesentlichen auf Gesamtsimulationen, die die einzelnen Programme zu Synchronisationszeitpunkten, deren Abstände konstant bleiben, mittels parallelem *loose coupling* kommunizieren lassen. Auf die genaue Vorgehensweise bei der Co-Simulation von BCVTB sei nun noch etwas näher eingegangen. Die Informationen hierzu sind [Wet11], Kapitel 4, entnommen.
Zunächst seien erneut zwei gekoppelte Simulatoren betrachtet und es werde angenommen, dass jedes Einzelsystem durch eine gewöhnliche Differentialgleichung beschrieben wird, für dessen Lösung jeweils die Zustände aus dem anderen System benötigt werden.

$$x_1(t_{k+1}) = f_1(x_1(t_k), x_2(t_k)) \tag{3.18}$$
$$x_2(t_{k+1}) = f_2(x_2(t_k), x_1(t_k)) \tag{3.19}$$

$x_i(t_k), \quad i \in \{1, 2\}$ steht hierbei für den Zustandsvektor im jeweiligen System zum k-ten Synchronisationszeitpunkt t_k, wobei $k \in \{0, \ldots, N-1\}$ bei insgesamt N Synchronisationszeitschritten. Der Zustand des Vektors wird dabei durch eine vom Solver der einzelnen Systeme abhängigen Funktion $f_i, \quad i \in \{1, 2\}$ zu jedem benötigten Zeitschritt berechnet. Da in den Teilsimulationen selbst kleinere als auch variable Zeitschritte verwendet werden können, solange die Kommunikation zu den vom Gesamtsolver gewünschten Zeiten gewährleistet ist, wird in diesen Zwischenschritten für die Zustände des anderen Systems schlichtweg der Wert bei der letzten Synchronisation für die Berechnung verwendet; es erfolgt demnach konstante Extrapolation. Die Reihenfolge des Aufrufs der Einzelsysteme ist somit für den Datenaustausch unerheblich.
Für die allererste Synchronisation müssen Anfangswerte gegeben sein, da sonst eine Schleife auftreten würde. Diese Werte werden bei der allerersten Synchronisation übergeben, danach erfolgt bis zum nächsten Synchronisationszeitpunkt die unabhängige Simulation mit den individuellen Solveralgorithmen der Einzelsysteme.
Im Beispiel zweier gewöhnlicher Differentialgleichungssysteme der Form

$$\dot{x_1}(t) = h_1(x_1, x_2) \tag{3.20}$$
$$\dot{x_2}(t) = h_2(x_2, x_1) \tag{3.21}$$

mit den Anfangswerten $x_1(t_0) = x_{1,0}$ und $x_2(t_0) = x_{2,0}$ erhält man nach der Verwendung eines expliziten Euleralgorithmus das System

$$x_1(t_{k+1}) = x_1(t_k) + h_1(x_1(t_k), x_2(t_k))\Delta t \tag{3.22}$$
$$x_2(t_{k+1}) = x_2(t_k) + h_2(x_2(t_k), x_1(t_k))\Delta t \tag{3.23}$$

Δt repräsentiert hierbei den im Allgemeinen konstanten Zeitschritt zwischen zwei Synchronisationen. Zu jedem Synchronisationszeitpunkt erfolgt, wie bereits beschrieben, der Austausch von $x_1(t_k)$ und $x_2(t_k)$, sodass alle für die Berechnung von $x_1(t_{k+1})$ und $x_2(t_{k+1})$ benötigten Werte vorhanden sind.
Diese Art der Kopplung erfordert keine Iteration, welche mit dem betrachteten Co-Simulationstool BCVTB zurzeit noch nicht zu verwirklichen ist, wie auch experimentell in Kapitel 6.1 dieser Arbeit zu sehen sein wird.
Zu beachten ist, dass bei dieser Art der Co-Simulation nicht nur die Fehler der einzelnen Solveralgorithmen aufzusummieren sind, sondern durch die getätigten Zeitschritte der einzelnen Solver zwischen zwei Synchronisationszeitpunkten die als konstant angenommenen Eingangswerte der anderen Simulationen signifikante weitere Fehler zustande kommen. Diese können in einer Langzeitsimulation einen beträchtlichen Betrag annehmen, der zur Kopplung detaillierter physikalischer Systeme selbst durchaus mäßigen Ansprüchen nach Genauigkeit nicht gerecht werden kann.

KAPITEL 4

Beschreibung der verwendeten Software

4.1 MATLAB

Die Informationen zu diesem sowie den beiden folgenden Unterkapiteln wurden aus [Mat] gewonnen.
MATLAB (kurz für *Matrix Laboratory*) ist eine Programmierumgebung, die eine eigene Programmiersprache, einen eigenen Simulator mit verschiedenen Solvern als auch verschiedene Toolboxen mit sich bringt, wodurch unterschiedliche Modellbildungsansätze unterstützt werden.
Numerische Werte werden in MATLAB generell als Matrizen interpretiert, was viele Berechnungsvorgänge beschleunigt und oftmals die Verwendung von Schleifen ersetzen kann. Ein Skalar repräsentiert somit in MATLAB eine 1×1-Matrix. Zuweisungen erfolgen in der MATLAB-Programmiersprache durch ein einfaches Gleichheitszeichen. Die Deklaration von Variablen ist nicht notwendig, da bei erstmaliger Zuweisung an eine noch nicht vorgekommene Variable diese sowohl deklariert als auch initialisiert wird. Listing 4.1 zeigt zwei Möglichkeiten zur Deklaration sowie Initialisierung eines Zeilenvektors mit den Einträgen 5 bis 13, dessen Einträge einmal in einer Schleife und einmal durch den MATLAB-Befehl *sum* addiert werden. Die Überprüfung am Ende zeigt, dass beide Arten dasselbe Ergebnis liefern. Das doppelte Gleichheitszeichen bedeutet eine Überprüfung auf Gleichheit der beiden Seiten, wie sie gewöhnlich in if-Abfragen verwendet wird. Die Auswertung ergibt in diesem Fall somit eine boolesche Variable mit dem Wert 1 für *true*.

```
1 >> y = 5:13;
2 >> a = 0;
3 >> b = 0;
4 >> for i = 1:length(x)
5      a = a + x(i);
6    end
```

```
>> b = sum(y);
>> x - y

ans =

     0     0     0     0     0     0     0     0     0

>> a == b

ans =

     1
```

Listing 4.1: MATLAB Code zur Deklaration als auch Initialisierung von Vektoren und unterschiedlichen Summenbildung

Neben vielen vorimplementierten Funktionen zur Behandlung von Problemen aus der Linearen Algebra, die für Matrizen nahe liegen, wie die Berechnung von Inversen, Eigenwerten und die Lösung von Gleichungssystemen, können mit MATLAB auch Daten interpoliert oder Differentialgleichungen gelöst werden. Listing 4.2 zeigt den Code zur Erstellung eines Function Handles samt der Lösung der dadurch definierten Differentialgleichung durch die Verwendung des Solvers ode45, dem ein Runge-Kutta-Verfahren der Ordnung 4 zugrunde liegt.

```
>> f = @(t,y) [y(2)*y(3);-y(1)*y(3);-0.51*y(1)*y(2)];
>> [t,y] = ode45(f,[0 15],[0 1 1]);
```

Listing 4.2: MATLAB Code zur Erstellung eines Function Handles und der Lösung einer Differentialgleichung

Die Darstellung der Lösung zu allen ausgewerteten Zeitschritten kann durch einen *plot*-Befehl geschehen. Abgesehen von diversen Einstellungen für diesen Befehl ist es auch möglich, dreidimensionale Kurven oder Flächen zeichnen zu lassen.
Alle bisher angeführten Befehle wurden im so genannten *Workspace* eingegeben, wodurch sofort deren Auswertung erfolgt. Zur Erstellung von Skripts oder Funktionen bietet MATLAB einen eigenen Editor, der für das MATLAB-spezifische Syntax-Highlighting und Debugging äußerst hilfreich ist. Neben der möglichen Integration von Source Code in andere Programmiersprachen wie z.B C, C++ und Java ist es durch die vielen verschiedenen Toolboxen, die von Mathworks für MATLAB angeboten werden, möglich, mithilfe von MATLAB ohne eine weiteres Programm verschiedene Modellbildungsansätze zu verwenden und zu vergleichen. Zwei dieser Toolboxen, die in dieser Arbeit ebenfalls verwendet werden, sind in den nächsten beiden Abschnitten vorgestellt.

4.2 Simulink

Die Simulink-Toolbox von MATLAB erlaubt die graphische Modellierung von kontinuierlichen als auch diskreten Systemen. Simulinkmodelle können als Blockdiagramme erstellt werden, die Signalflussgraphen entsprechen. Standardmäßig sind in den Simulink-Bibliotheken Blö-

cke für mathematische Operationen wie Addition, Multiplikation oder Integration, Generatoren für diverse Signale als auch Übertragungsfunktionen, Filter und PID-Regler vorhanden. Zudem können einerseits selbst geschriebene MATLAB-Funktionen in Blöcken zu Simulinkmodellen hinzugefügt werden, andererseits ist es auch möglich, über MATLAB Simulinkmodelle aufzurufen und aus einer Simulink-Simulation gewonnene Daten in MATLAB weiterzuverarbeiten. Für die Simulation von Simulinkmodellen stehen eigene Solver zur Verfügung, deren Algorithmen von expliziten als auch impliziten Fixed-Step-Verfahren bis hin zu Verfahren mit variabler Schrittweite sowie Solvern, die speziell für steife oder gemischte Systeme geeignet sind, reichen.

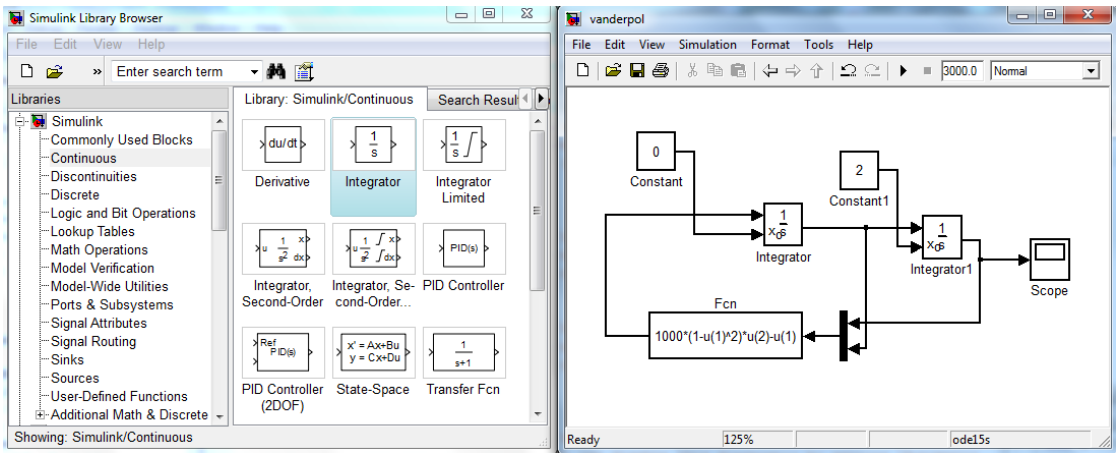

Abbildung 4.1: Simulink Library Browser und graphisches Modell der Differentialgleichung eines Van-der-Pol-Oszillators

Abbildung 4.1 zeigt den Simulink Library Browser sowie das Modell zur Lösung der Differentialgleichung

$$\ddot{y} = 1000(1 - y^2)\dot{y} - y;$$

für einen Van-der-Pol-Oszillator mit den Anfangsbedingungen

$$y(0) = 2$$
$$\dot{y}(0) = 0.$$

Neben dem Funktionsblock und den Integratoren, die für die Modellierung der Differentialgleichung benötigt werden, sind einerseits die zur Simulation notwendigen Anfangsbedingungen über die *Constant*-Blöcke angegeben, andererseits wird auch der Wert von y über alle Zeitschritte in einem *Scope*-Block visualisiert. Die Simulation kann entweder durch Betätigung des schwarzen Dreiecks erfolgen oder durch den Aufruf aus dem MATLAB-Workspace oder einem MATLAB-Skript mit dem Befehl *sim*:

```
>> [t,y]=sim('vanderpol');
```

Neben dem Namen des Modells können noch einige Parameter optional in diesem Befehl definiert werden, wie der verwendete Solver und dessen maximale Schrittweite oder die Start- sowie Endzeit der Simulation. Durch das Voranstellen von $[t, y] =$ wird in der Variablen t die Simulationszeit gespeichert und in y die Lösung der Differentialgleichung zu jedem Zeitschritt. Durch einen Doppelklick auf den *Scope*-Block nach Ablauf der Simulation erhält man die in Abb. 4.2 abgebildete Graphik, die die Lösung y zu jedem berechneten Zeitschritt zeigt.

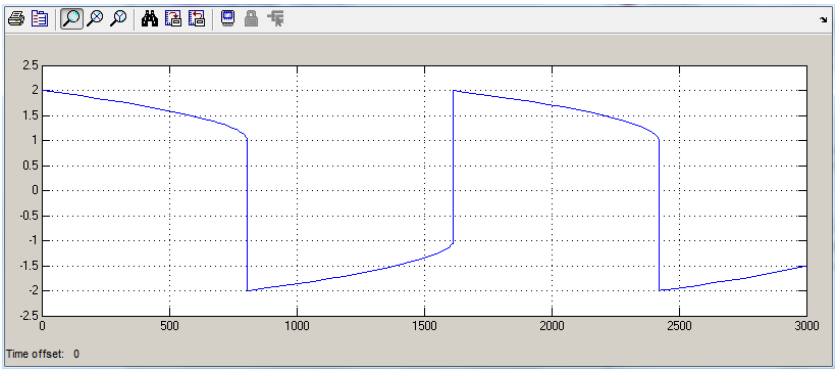

Abbildung 4.2: Über einen Scope-Block geplottetes Signal zu jedem ausgewerteten Zeitschritt

4.3 Simscape

Die MATLAB-Toolbox Simscape ermöglicht graphische akausale Modellbildung physikalischer Systeme aus unterschiedlichen Domänen. Simscape-Modelle sind gleichungsbasiert, was bedeutet, dass die Verbindungen einzelner Blöcke nicht gerichtet sind, sondern immer die beiden Größen *effort* und *flow* (in Simscape *across* bzw. *through* genannt) mit sich bringen, deren Wert zu jedem Zeitpunkt aus den aus dem gesamten System hervorgehenden Gleichungen berechnet wird, ohne dass von Beginn an feststeht, welche der beiden Variablen von der anderen abhängt (siehe auch Kapitel 2.2). Das Simscape-Modell für einen Schwingkreis ist in Abb. 4.3 gezeigt.

Das Modell entspricht grundsätzlich dem zugehörigen Schaltbild, notwendig ist außerdem ein Erdungs-Block (*electrical reference*) sowie ein *solver configuration*-Block, der es möglich macht, in komplexen Modellen für einzelne Teile einen eigenen, anderen Solver zu verwenden als für die Simulation des gesamten Modells, was einer Multirate-Simulation entspricht (siehe Kapitel 3). Wird kein separater Solver benötigt, kann in den Einstellungen dieses Blocks angegeben werden, dass der globale Solver benützt werden soll; es ist jedoch nicht möglich, den Block einfach wegzulassen.
Wie bereits erwähnt, werden die Werte der *effort*- und *flow*-Variablen an jeder Position für alle Zeitschritte aus Gleichungen berechnet, die die jeweiligen Blöcke definieren. Teile des Simscape-Codes für den Kondensator-Block sind in Listing 4.3 angeführt.

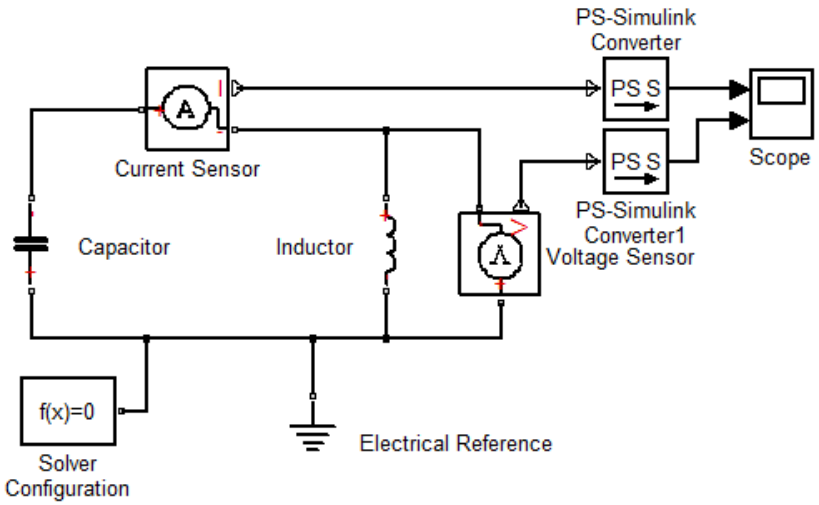

Abbildung 4.3: Simscape-Modell eines idealen Schwingkreises

```
component capacitor < foundation.electrical.branch

    %%-------------------------------------------------------------------------%%
    %%-->>Beschreibung des Kondensator-Blocks<<-----------------------------%%
    %%-------------------------------------------------------------------------%%
% Copyright 2005-2008 The MathWorks, Inc.

  parameters
    c  = { 1e-6, 'F' };    % Capacitance
    v0 = { 0, 'V' };       % Initial voltage
    r  = { 1e-6, 'Ohm' }; % Series resistance
    g  = { 0, '1/Ohm' };   % Parallel conductance
  end

  variables
    vc = { 0, 'V' }; % Internal variable for voltage across capacitor term
  end

  function setup
    if c <= 0
        pm_error('simscape:GreaterThanZero','Capacitance')
    end
    if g < 0
        pm_error('simscape:GreaterThanOrEqualToZero','Parallel conductance')
    end
    if g == {inf, '1/Ohm'}
        pm_error('simscape:LessThan','Parallel conductance','inf')
    end
    if r < 0
        pm_error('simscape:GreaterThanOrEqualToZero','Series resistance')
    end
```

```
32      if r == {inf, 'Ohm'}
33          pm_error('simscape:LessThan','Series resistance','inf')
34      end
35      vc = v0; % Assign initial voltage
36    end
37
38    equations
39      v == i*r + vc;
40      i == c*vc.der + g*vc;
41    end
42
43  end
```

Listing 4.3: Codesequenzen aus dem Simscape-Code für die Kondensator-Komponente, Copyright: The MathWorks [Mat]

Die erste Zeile zeigt, dass *capacitor* von der Klasse *foundation.electrical.branch* erbt. Unter *parameters* werden alle Parameter samt deren physikalischer Einheit und dem Default-Wert angegeben. Variablen, die nur intern benötigt werden, können unter *variables* deklariert werden. Nach dem Code zur Vergewisserung der Eingabe physikalisch sinnvoller Werte für die einzelnen Parameter kommt der *equations*- Abschnitt, in dem die Gleichungen, die die jeweilige Komponente repräsentieren, angegeben werden. Es ist zu beachten, dass es sich hierbei tatsächlich um Gleichungen, die erfüllt werden müssen, handelt und nicht um Zuweisungen, was durch das doppelte Gleichheitszeichen realisiert ist. Im Gegensatz dazu handelt es sich bei dem einfachen Gleichheitszeichen in Zeile 35 tatsächlich um eine Zuweisung, die deshalb außerhalb des Gleichungsabschnitts programmiert werden muss.
Um auf die *effort*- und *flow*-Variablen (in diesem Fall Spannung und Stromstärke) an gewissen Stellen zugreifen zu können, müssen Sensoren-Blöcke eingefügt werden. In dem Modell aus Abb.4.3 wird die an der Spule anliegende Spannung abgegriffen und der Stromfluss vom Kondensator zur Spule gemessen. Die Messblöcke verfügen über einen Ausgang, an dem man ein gerichtetes, jedoch physikalisches Signal, also mit einer physikalischen Einheit versehen, erhält. Dieses kann über einen eigenen Umwandlungsblock in ein einheitenloses Simulink-Signal konvertiert und somit in einem Simulinkmodell oder einzelnen Simulinkblöcken, wie hier in einem Scope-Block, weiterverwendet werden.
Simscape-Modelle können ebenso wie Simulink-Modelle über MATLAB aufgerufen und deren Daten weiterverarbeitet werden.

4.4 Dymola/Modelica

Bei der Installation von Dymola [Sys] erhält man sowohl einen Simulator als auch eine graphische Modellbildungsumgebung für die objektorientierte Programmiersprache Modelica [Assb], die speziell für die gleichungsbasierte Modellbildung physikalischer Systeme entwickelt wurde. Modelica selbst ist nicht kommerziell, die Modelica Standard Library sowie viele andere Bibliotheken werden direkt auf der Modelica-Homepage zum kostenlosen Download angeboten. Zur Simulation von Modelica-Modellen wird jedoch ein Simulator benötigt, der nicht direkt über die Modelica Association erhalten werden kann. Ein weit verbreiteter kos-

tenloser Simulator, der auch eine Modellbildungsumgebung beinhält, wäre OpenModelica [Con]. Kommerzielle Programme, die den Modelica-Standard verwenden, sind z.B. MapleSim [Map], SimulationX [ITI], Wolfram SystemModeler [Wol] oder die in dieser Arbeit verwendete Umgebung Dymola. Dymola wurde insbesondere deshalb gewählt, weil die Kommunikation des Co-Simulationstools BCVTB mit diesem Modelica-Simulator am besten dokumentiert ist. Ein weiterer Vorteil von Dymola gegenüber anderen Simulatoren ist jener, dass für gewöhnlich Dymola am schnellsten an neue Modelica-Versionen angepasst wird.
Wie bereits erwähnt, sind Modelica-Modelle gleichungsbasiert; in Dymola können diese Modelle sowohl graphisch als auch textuell erstellt und bearbeitet werden. Während des gesamten Modellierungsvorgangs kann zwischen diesen beiden Ansichten einfach in der Menüleiste umgeschaltet werden. Abb. 4.4 zeigt diese beiden Ansichten des Modells eines Schwingkreises.

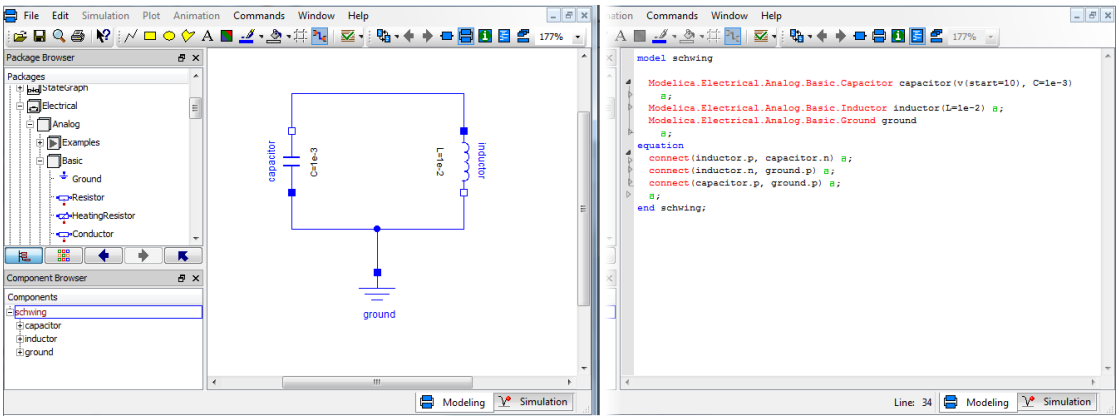

Abbildung 4.4: Dymola-Modell eines idealen Schwingkreises, textuell als auch graphisch

Die *connect*-Befehle im Gleichungsabschnitt beinhalten die Gleichungen, die zur Erfüllung der Kirchhoffschen Gesetze benötigt werden. Für

```
connect(inductor.p, capacitor.n);
```

bedeutet dies

```
inductor.p.v = capacitor.n.v;
inductor.p.i - capacitor.n.i = 0;
```

wobei über die Endung v auf die *effort*-Variable (in diesem Fall die Spannung) zugegriffen wird und über die Endung i auf die *flow*-Variable (in diesem Fall die Stromstärke). Zu beachten ist, dass in Modelica Gleichungen im Gegensatz zu Simscape über ein einfaches Gleichheitszeichen realisiert werden, Zuweisungen hingegen durch ein $:=$. Ein weiterer Unterschied, der hinsichtlich der Komplexität großer Systeme eine nicht unbedeutende Rolle spielt, ist die Tatsache, dass keine Sensoren in das Modell eingebaut werden müssen, um nach einer Simulation auf alle gewünschten Werte zugreifen zu können. Um ein mit Dymola erstelltes Modell zu simulieren und Solver-Einstellungen zu konfigurieren, kann in der unteren Leiste auf das *Simulation*-Fenster geklickt werden. Nach der Definition der Start- und

Endzeit sowie des Solvers kann durch einen Klick auf den *Simulate*-Button, dessen Icon jenem unten neben dem Fenstertitel gleicht, die Simulation gestartet werden. Sofern die Simulation erfolgreich war, erscheint im linken Fenster eine Liste aller im Modell verwendeten Komponenten (siehe Abb. 4.5).

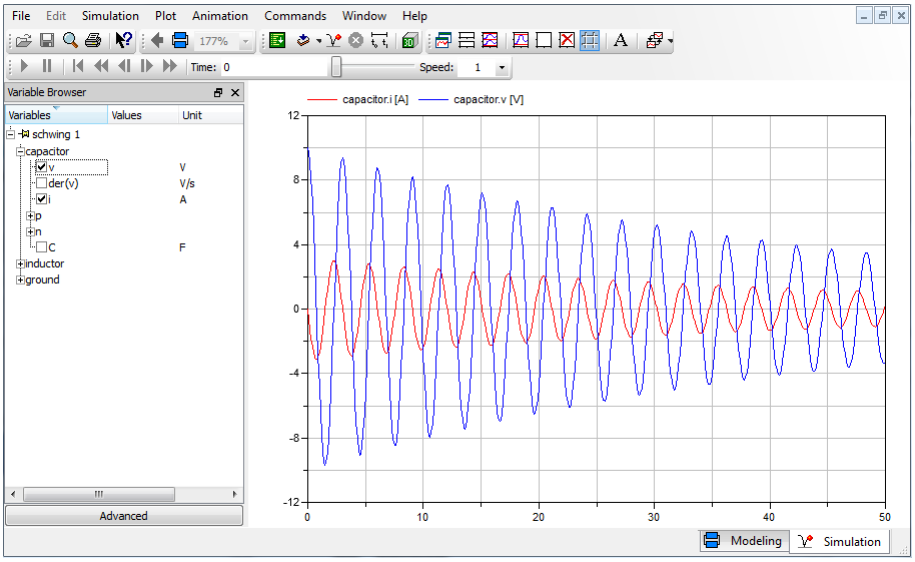

Abbildung 4.5: Simulationsfenster und -ergebnis für den Schwingkreis

Für jede Komponente kann man einerseits auf das Potential und die Stromstärke an jedem Verbindungsknoten als auch direkt auf die an der Komponente anliegende Spannung, die automatisch aus der Potentialdifferenz an den beiden Knoten berechnet wird, oder den Stromfluss, der schlichtweg vom p-Knoten übernommen wird, zugreifen. Sobald eine dieser Variablen aus der Liste angeklickt wird, erscheint deren Verlauf über die Simulationszeit im Plot rechts davon. Auf diese Weise kann jede beliebige Variable rasch und auf unkomplizierte Weise beobachtet werden.
Ähnlich, wie es bei MATLAB und Simulink der Fall war, können auch Dymolamodelle über ein Skript ausgeführt werden, dessen Dateiname in diesem Fall die Endung *.mos* besitzt. Die Simulation einen Dymolamodells erfolgt über den Befehl *simulateModel*:

```
simulateModel("schwing", startTime=0, stopTime=50, method="Dassl");
```

Alle Parameter nach dem Dateinamen des Modells sind optional und überschreiben gegebenenfalls existente Simulationseinstellungen für die Dauer dieser Simulation.

4.5 EnergyPlus

Die Gebäudesimulationssoftware EnergyPlus ist ein eigens auf die Ermittlung des Energieverbrauchs von Gebäuden ausgerichtetes Programm, das an der University of California und der University of Illinois entwickelt wurde. Die Entwickler stellen deutlich klar, dass die

Freeware EnergyPlus kein User Interface ist, dennoch kann jede EnergyPlus-Datei einerseits mit jedem Textverarbeitungsprogramm als auch mit dem bei der Installation von EnergyPlus inkludierten IDF-Editor bearbeitet werden. Die Simulation kann über das ebenfalls mitgelieferte EPLaunch ausgeführt werden. Über den IDF-Editor können Objekte aller in EnergyPlus verfügbaren Klassen erstellt und deren Parameter modifiziert werden. Trotz der allgemein spärlich gegebenen Übersichtlichkeit hat der IDF-Editor im Gegensatz zu einem Texteditor den Vorteil, dass auch Klassen ohne bereits vorhandene Elemente sichtbar sind und im Falle von Einschränkungen für bestimmte Parameter diese im IDF-Editor angegeben werden.
Um ein EnergyPlus-Modell zu erstellen, muss neben den zu verwendenden Simulationsparametern zunächst die gesamte Geometrie des Gebäudes angegeben werden. Für ein EnergyPlus-Modell wird das Gebäude in Zonen unterteilt, die ähnliche thermische Rahmenbedingungen repräsentieren. So kann beispielsweise für jeden Raum eine Zone erstellt werden, andererseits ist es auch möglich, einige Räume zu einer Zone zusammenzufassen, falls das thermische Verhalten dieser Räume als für die Bestimmung des Energieverbrauchs hinreichend gleich angesehen werden kann. Einen Raum in mehrere Zonen zu unterteilen, liegt offenbar nicht im Interesse der Entwickler, da jede Zone an allen Seiten von Oberflächen begrenzt sein muss. Würde eine dieser Wände zwischen zwei Zonen mit minimalem thermischen Widerstand bedacht, vernachlässigte diese Methode dennoch die natürliche Bewegung der Luft, die lediglich durch künstlich eingebrachte Luftströme und damit verbundenen unverhältnismäßig großen Aufwand im Modell berücksichtigt werden könnte.
Neben der genauen Geometrie des Gebäudes unter Verwendung diverser vor- oder vom Benutzer definierter Materialen sowie eines Koordinatensystems zur Festlegung aller benötigten Knoten können jeder Zone Wärmequellen aus Lichtern, Personen oder Maschinen, luft- oder wasserbasierte Kühlsysteme als auch äußere Bedingungen wie Sonneneinstrahlung, Außentemperatur, Windstärke und Bewölkungsgrad zugeordnet werden.
Für die Simulation eines EnergyPlus-Modells wird zunächst ein Zeitschritt festgelegt, der in Bruchteilen einer Stunde angegeben wird; der kleinstmögliche Wert entspricht einer Minute. Dieser Zeitschritt wird für den Algorithmus, der für die Berechnung der thermischen Vorgänge in einer Zone bestimmt wird, verwendet. Diese Vorgänge werden durch eine Differentialgleichung für die Temperaturänderung beschrieben, die Wärmezu- und -abflüsse sowie Oberflächen und Wärmeübergangskoeffizienten berücksichtigt. Details hinsichtlich dieser Gleichung und deren Vereinfachung seien vom interessierten Leser der EnergyPlus-Dokumentation [Lab] entnommen. Zur Lösung dieser Wärmebilanzgleichung kann zwischen den Algorithmen *ThirdOrderBackwardDifference*, *EulerMethod* und *AnalyticalSolution* gewählt werden. Die ersten beiden Algorithmen verwenden numerische Finite Differenzen Methoden dritter bzw. erster Ordnung, *AnalyticalSolution* berechnet eine analytische Lösung. Um bestimmte Konvergenzkriterien einzuhalten oder die Werte für den Austausch mit Verfahren für den Übergang zwischen den einzelnen Zonen sowie für den Luft- und Wassertransport der Kühl- bzw. Heizungssysteme hinreichend genau zu ermitteln, kann das System auch kleinere Zeitschritte verwenden, von denen der Benutzer während einer Simulation nicht in Kenntnis gesetzt wird. Allgemein wird für die einzelnen Bereiche stets ein Predictor-Corrector-Verfahren verwendet. Der interne Austausch erfolgt nach dem Gauß-

Seidel-Prinzip, welches unter anderem in Kapitel 3 erklärt wird.
Als weiterer interessanter Aspekt hinsichtlich der Simulationsergebnisse von EnergyPlus ist zu bemerken, dass die Anfangstemperaturen der verwendeten Oberflächen nicht vom Benutzer eingestellt werden können. Diese sind von EnergyPlus vorgegeben und ermöglichen eine Konvergenz, um den angegebenen Umgebungsbedingungen zu entsprechen, innerhalb der so genannten *warmup days*. Falls vom Benutzer zu wenige dieser *warmup days* erlaubt werden, kann es geschehen, dass das System nicht innerhalb dieser Zeit konvergiert, was eine vermeintliche Verfälschung der Startwerte zur Folge hat. Die Informationen dieses Kapitels wurden aus [Lab] zusammengefasst.

4.6 BCVTB

Das Building Controls Virtual Test Bed, abgekürzt BCVTB, ist ein an der University of California entwickeltes Tool, das speziell für den Zweck der Co-Simulation erstellt wurde. Die Informationen dieses Kapitels entstammen der von Michal Wetter verfassten Dokumentation ([Wet12]).
BCVTB basiert auf Ptolemy II [otUoC], einem Programm zur graphischen Modellierung unterschiedlicher Systeme, die in *Actor*-basierten Graphen dargestellt werden. Grundsätzlich sind Actors in Ptolemy schlichtweg Blöcke mit Ein- und Ausgängen, sodass bei deren Verbindung Signalflussgraphen entstehen. Die Simulation eines Ptolemy-Modells wird durch so genannte *Directors* gesteuert, die auf die Art der benötigten Simulation (z.B. Process Networks, Discrete Event, Continuous-Time etc.) zugeschnitten sind. Ptolemy-Modelle können hierarchisch angeordnet werden, was bedeutet, dass in einem Ptolemy-Modell mehrere Untermodelle verschiedener Stufen auftreten können. In jedem Untermodell kann ein eigener Director verwendet werden, was Multirate-Simulation ermöglicht.
Die BCVTB-Oberfläche scheint zunächst jener von Ptolemy zu gleichen, allerdings kann man beim der Betrachtung der Actors feststellen, dass BCVTB nur über einen Bruchteil der Ptolemy-Actors verfügt, jedoch im Gegenzug einige neue Actors mit sich bringt, die speziell für Co-Simulation benötigt werden. Einige dieser Actors werden in Kapitel 5.1 noch genauer vorgestellt. Um mithilfe von BCVTB mehrere Simulatoren zu koppeln, wird zunächst ein BCVTB-Modell erstellt, das zumindest einen Director und für jedes Modell, das gekoppelt simuliert werden soll, einen *Simulator*-Actor beinhalten muss. Jeder dieser Actors bewirkt bei der Simulation des Modells die Öffnung einer Instanz des jeweiligen Programms und die Ausführung eines zugehörigen Skripts, das entweder selbst den Source Code für das entsprechende Modell enthält oder im Falle eines Blockdiagrammmodells dazu benötigt wird, um dieses aufzurufen. Abbildung 4.6 zeigt ein Beispiel eines BCVTB-Modells zur Co-Simulation von MATLAB und Dymola.

Da BCVTB per se nicht wissen kann, welche Dimensionen das Ein- und Ausgangssignal für die einzelnen Simulatoren haben muss, werden die Signale durch einen *VektorAssembler* für die Simulator Actors zu einem Vektor zusammengefasst und zur Weiterverwendung in BCVTB wieder umgewandelt. Der Director (in diesem Fall ein *Synchronous Data Flow*-Director, der in Kapitel 5.1 genauer beschrieben wird) steuert die Simulation des Gesamtmodells und bewirkt so eine Kommunikation aller Simulatoren zu den von ihm gesetzten

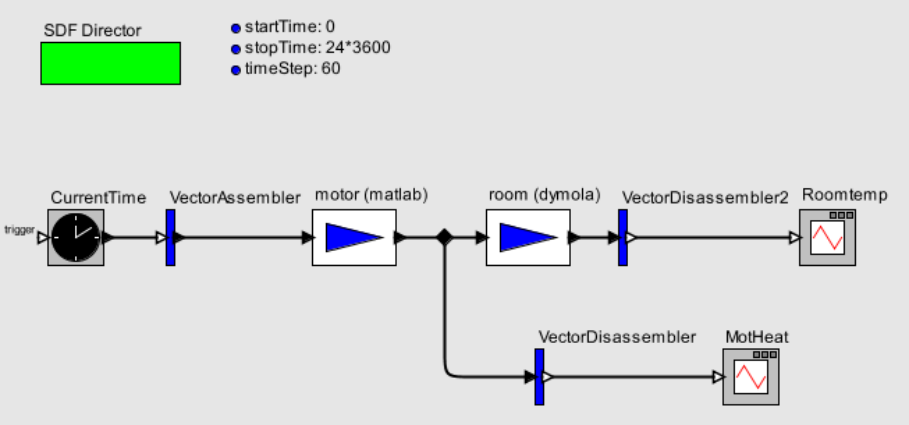

Abbildung 4.6: BCVTB-Modell zur Kopplung einer Dymola- und einer MATLAB-Instanz

Zeitschritten.
Die Kommunikation der verschiedenen Programme erfolgt über so genannte BSD-Sockets (kurz für engl. *Berkeley Software Distribution Sockets*). Diese Sockets wurden ebenfalls an der University of California für die Interprozesskommunikation entwickelt. Für eine Co-Simulation mit BCVTB muss von jedem gekoppelten Programm ein Socket für den zugehörigen Prozess erstellt werden, durch welchen der Datenaustausch mit anderen Prozessen überhaupt ermöglicht wird. Hierbei stellt der von BCVTB gestartete Prozess den so genannten *Server* dar, während jeder weitere, über Sockets mit diesem Prozess kommunizierende Prozess, als *Client* bezeichnet wird. Der Server wartet zu jedem gewünschten Zeitpunkt darauf, dass von den Clients Daten rücktransferiert werden und meldet einen Fehler, falls der Client keine Daten liefert oder zu einem nicht vorgesehenen Zeitpunkt Daten zurückschicken möchte. Falls beim Client keine Daten vom Server zur Verfügung stehen, wenn sie gebraucht werden, führt dies meist zu einem programminternen Fehler und einer daraus resultierenden Terminierung dieses Programms.
In dieser Arbeit wird lediglich die Verbindung von Prozessen, die auf einem Computer mit 4 Prozessoren laufen, betrachtet; durch BSD-Sockets wäre es jedoch generell möglich, auch zwischen Prozessen, die an verschiedenen Maschinen laufen, eine Kommunikation aufzubauen. Nähere Informationen über BSD-Sockets können [SFR04] entnommen werden.
Bei der Installation von BCVTB ist darauf zu achten, dass alle Versionen der über BCVTB zu koppelnden Programme kompatibel sind, da es sonst zu Fehlermeldungen kommt, die nicht auf diese Ursache schließen lassen.

KAPITEL 5

Modellbeschreibung und Implementierung

In Anlehnung an das in Kapitel 1.1 vorgestellte INFO-Projekt ist ein Ziel dieser Arbeit, anhand von Beispielen der Kopplung von Maschinen- und Raummodellen die Möglichkeiten der Co-Simulation mit BCVTB zu erörtern. Hierzu ist es zunächst notwendig, die Synchronisation der verwendeten Software zu ermöglichen und so den von BCVTB angesteuerten Datenaustausch zu verwirklichen. Mit den hierzu notwendigen Modifikationen beschäftigt sich der erste Teil dieses Kapitels.
Zur Erstellung eines Raummodells in Dymola mussten zunächst einige Komponenten entwickelt werden, um mit deren Hilfe jeden der in den Experimenten benötigten Räume modellieren zu können. Die Ausgangsbasis für diese thermischen Compartments sowie deren Erweiterung zur Berücksichtigung von Konvektion sind im Abschnitt 5.3 erklärt. Für die Maschinen sowie die Temperaturregelung werden Modelle in unterschiedlichen Modellierungsumgebungen herangezogen, weshalb versucht wird, neben der Beschreibung der jeweiligen Modelle auch auf deren Unterschiede hinsichtlich der Modellbildungsart hinzuweisen.

5.1 Synchronisation

Die Co-Simulation der Einzelmodelle in verschiedenen Simulatoren erfolgt in dieser Arbeit stets über das Building Controls Virtual Test Bed. Um die Gesamtsimulation zu steuern, müssen zum einen die einzelnen Programme von BCVTB angesteuert werden und diese Programme während der Simulation wiederum Daten an BCVTB liefern, zum anderen muss zwischen diesen gegenseitigen Aufrufen die Kommunikation ermöglicht werden, was über BSD-Sockets geschieht. Diese Sockets wurden an der University of California als Programmierschnittstelle für Interprozesskommunikation entwickelt. Detaillierte Informationen über diese Sockets können Kapitel 4.6 bzw. [SFR04] entnommen werden.
Zur Simulation des Gesamtmodells bietet BCVTB drei verschiedene Directors, die den Sol-

ver bestimmen, welcher die Kommunikation aller verwendeten Simulatoren steuert. Der *Synchronous Data Flow* (SDF) Director arbeitet mit fixer Schrittweite, zu der alle verwendeten Simulatoren mit dem System kommunizieren müssen. Die modifizierbaren Parameter können Abb. 5.1 entnommen werden.

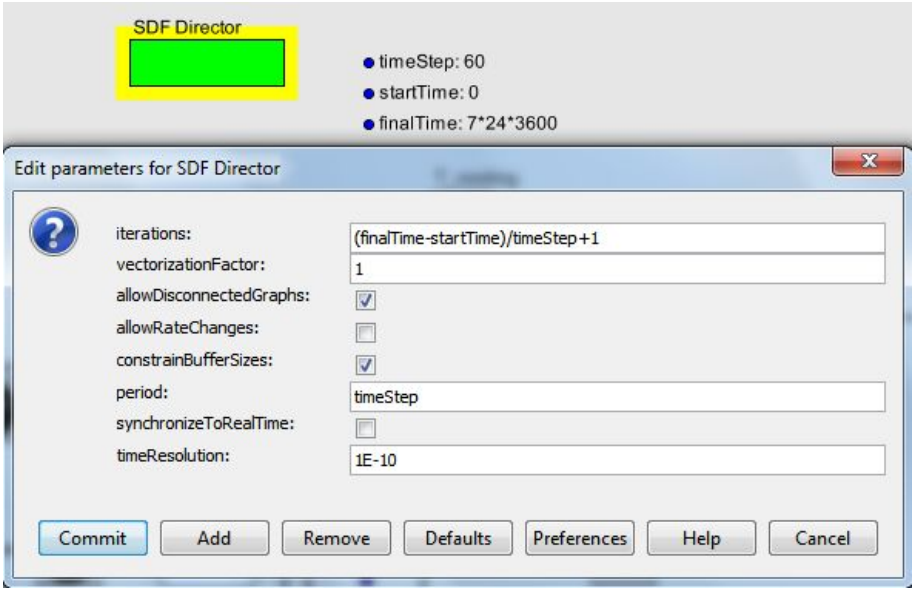

Abbildung 5.1: Parameter und Optionen des SDF Directors

Der erste zu setzende Parameter definiert die Anzahl an Iterationen, die der SDF Director vornehmen soll. Da für gewöhnlich nicht die Anzahl der Iterationen, sondern die Simulationsdauer von Interesse ist, wird in diesem Beispiel die Anzahl aus den vom Benutzer definierten Parametern für den Beginn und das Ende der Simulationszeit sowie der Schrittweite (*finalTime*, *startTime* bzw. *timeStep*) berechnet. Die Angabe einer endlichen Iterationsanzahl ist zudem wichtig, um ein ewiges Weiterlaufen der Simulation zu verhindern, falls kein fehler- oder zeitbedingter Abbruch einer Einzelsimulation die Gesamtsimulation beenden würde. Die als Wert von *timeStep* gewünschte Synchronisationsschrittweite erreicht man durch das Setzen des Parameters *period*. Um Echtzeitsimulationen zu ermöglichen, kann der Parameter *synchronizeToRealTime* aktiviert werden.
In den meisten der bei der Installation von BCVTB mitgelieferten, vorimplementierten Beispiele wird der SDF Director verwendet. Die einzelnen über ein BCVTB-Modell gekoppelten Simulatoren verwenden zudem einen Fixed-Step-Solver mit derselben Schrittweite, die im SDF Director gewählt ist. In diesem Fall verläuft zwar die Synchronisation problemlos ohne diesbezügliche Modifikation der Einzelmodelle, die Verwendung derselben Schrittweite in jedem Simulator ist für den Grundgedanken der Co-Simulation jedoch unzureichend. Wie bereits in Kapitel 3 beschrieben, wird Co-Simulation weitgehend dazu gebraucht, um Modelle zu kombinieren, deren Solver aufgrund der unterschiedlichen Anforderungen der einzelnen Systeme signifikant von einander abweichende Schrittweiten aufweisen. Demnach wird die

Verwendung derselben Schrittweite zur Simulation jedes Teilmodells im Allgemeinen nicht in Betracht gezogen.
Falls das Gesamtmodell einen Solver mit Schrittweitensteuerung benötigt, kann ein *Continuous Director* verwendet werden. Im Gegensatz zum SDF Director, der Synchronisation in gleichen Zeitabständen fordert, werden die Synchronisationszeitpunkte beim Continuous Director von einem Differentialgleichungssolver festgelegt. Wie Abb. 5.2 veranschaulicht, kann bei Verwendung eines Continuous Directors zur Simulation des Gesamtmodells zwischen den Solvern *ExplicitRK23Solver* und *ExplicitRK45Solver* gewählt werden, welche beide auf einem expliziten Runge-Kutta-Verfahren basieren. Zudem kann das Verhalten des Solvers durch Parameter wie die Simulationsstart- und -endzeit, die maximale Anzahl an Iterationen, die maximale Fehlertoleranz als auch die maximale Schrittweite sowie jene für den ersten Zeitschritt beeinflusst werden.

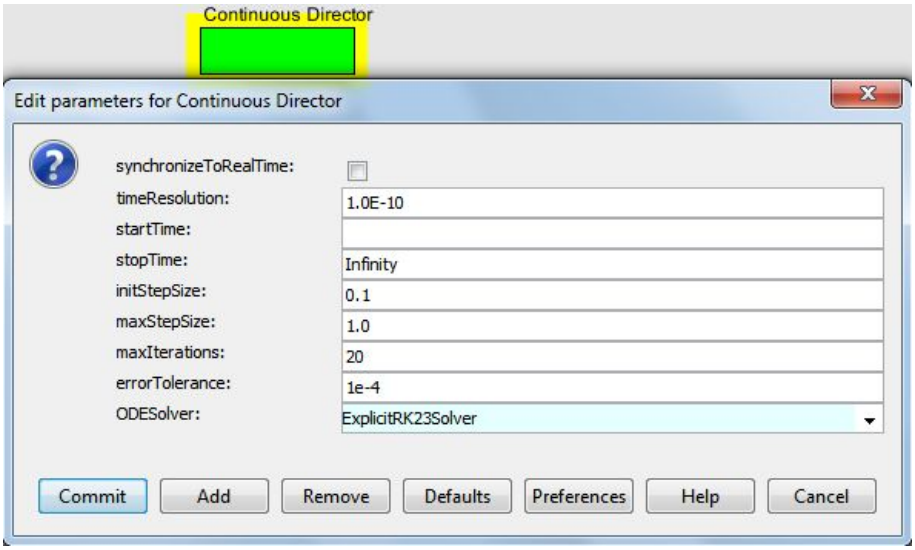

Abbildung 5.2: Parameter und Optionen des Continuous Directors

Die letzte Option zum Synchronous Data Flow Director und dem Continuous Director stellt der *FSM Director* dar. FSM steht für *Finite State Machine*, was übersetzt Zustandsautomat bedeutet. Der FSM Director besitzt nur einen Inputparameter, welcher ebenjenen Zustandsautomaten bezeichnet, der die Simulation steuern soll. Dieser Automat kann als selbst programmierter Actor auftreten, was eine komplett individuell gesteuerte Kommunikation ermöglicht. Da bereits der Continuous Director für Dymola- und EnergyPlusmodelle unüberbrückbare Hürden aufweist, wurde in dieser Arbeit der FSM Director in keinem der Experimente berücksichtigt, weshalb an dieser Stelle nicht weiter auf ihn eingegangen wird.
Weitere speziell für die Co-Simulation entwickelte Actors von BCVTB stellen die *Simulator Actors* dar. Diese Actors repräsentieren den anzusteuernden Simulator als Block im BCVTB Modell. Alle Signalflüsse, die in einen Simulator Actor hineinführen, werden zu jedem Synchronisationszeitpunkt an den gewählten Simulator übergeben; alle Signalflüsse von einem Simulator Actor stellen die in jedem Synchronisationszeitschritt vom jeweiligen Simulator

erhaltenen Werte dar. Die Parameter eines Simulator Actors können Abb. 5.3 entnommen werden.

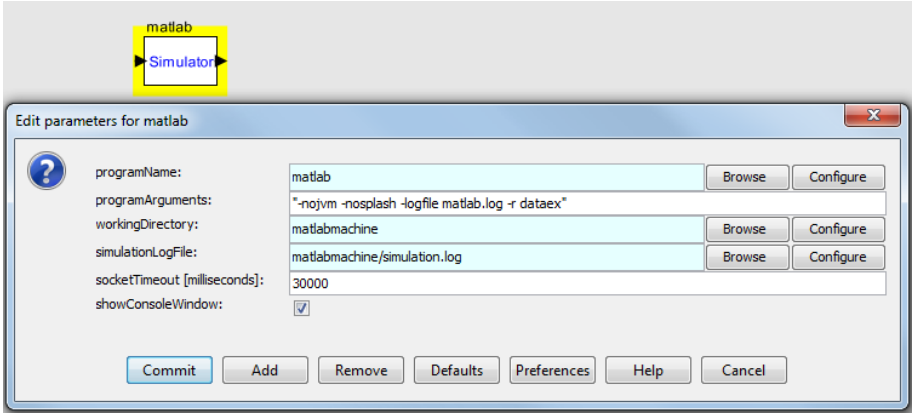

Abbildung 5.3: Parameter und Optionen des Simulator Actors

Der erste einzustellende Parameter *programName* definiert den Simulator. In *programArguments* können optional Einstellungen wie das Unterdrücken graphischen Outputs oder reine Leserechte festgesetzt werden. Die letzte, notwendige Option in dieser Zeile beschreibt den Namen der Script-Datei, die mit der in *programName* gewählten Software ausgeführt wird. Sowohl für die Kommunikation mit MATLAB als auch mit Dymola werden Scripts verwendet. Diese .m bzw. .mos - Dateien beinhalten folglich den Code zur Ausführung von Simulink- oder Simscape- bzw. Dymolamodellen. Die beiden nächsten Parameter des Simulator Actors definieren den Dateipfad, an dem sich die auszuführenden Dateien befinden, und den Dateinamen sowie den Ort, an dem das Log-File gewünscht wird. Der Parameter *socketTimeout* bestimmt die Anzahl an Millisekunden, die BCVTB auf die Antwort des jeweiligen Simulators warten soll. Diese Zeitspanne muss zum einen groß genug gewählt werden, um dem Programm genügend Zeit zum Hochfahren zu ermöglichen, andererseits führt ein zu hohes socketTimeout dazu, dass bei möglichen Fehlern im Programm oder der Kommunikation dennoch so lange nicht abgebrochen wird, bis die festgelegte Zeit abgelaufen ist.
Bereits zu Beginn dieses Kapitels wurde darauf hingewiesen, dass die Kommunikation zwischen den einzelnen Simulatoren mittels BSD-Sockets erfolgt. Die Erstellung dieser Sockets erfolgt je nach Software durch von BCVTB zur Verfügung gestellte Funktionen oder eigens konzipierte Blöcke, die den Code zum Aufbau der Verbindung über BSD Sockets enthalten.

5.1.1 Kommunikation mit Dymola

Der Aufruf eines Dymolamodells für die Co-Simulation mit BCVTB erfolgt, wie soeben erwähnt, mittels eines .mos-Scripts, dessen Name dem zugehörigen Simulator Actor als Parameter übergeben wird. Dieses Script beinhaltet lediglich den Code zum Simulationsaufruf des Dymolamodells (siehe Listing 5.1); die Erstellung des Sockets findet sich im sogleich beschriebenen BCVTB-Block.

```
1 simulateModel("dymolaroom", startTime=0, stopTime=7*24*3600, method="Dassl",
2 tolerance=1e-3);
3 exit();
```

Listing 5.1: Source Code eines .mos-Scripts zur Ausführung eines Dymolamodells

Der für Dymola entworfene BCVTB-Block bietet die Möglichkeit zur Definition des Zeitschritts, zu dem der Solver für die Simulation des Dymolamodells mit der BCVTB-Umgebung kommunizieren soll (siehe Abb. 5.4). Diese beim ersten Anblick sehr praktisch erscheinende Option impliziert sofort, dass Dymola nicht zu variablen Zeitschritten feuern kann, was bedeutet, dass für die Gesamtsimulation des BCVTB-Modells kein Continuous Director verwendet werden kann, sobald ein Dymolamodell mit eingebunden wird. In diesem Fall muss demnach ein SDF Director verwendet werden, dessen fixe Schrittweite als Parameter *timeStep* im BCVTB-Block des Dymolamodells verwendet werden muss. Die Schrittweite des in Dymola verwendeten Solvers kann hingegen separat gewählt werden oder auch variabel sein. Dymola feuert in jedem Fall zu dem Zeitschritt, der im BCVTB-Block eingestellt wurde und dem des SDF-Directors im BCVTB-Modell entspricht; die Zeitschritte in Dymola selbst umfassen somit diese sowie die für die Berechnung notwendigen, vom Solver in Dymola vorgegebenen Schritte. An dieser Stelle sei auf die Resultate der Experimente in Kapitel 6.1 verwiesen, wo auf die dokumentierten Schrittweiten genauer eingegangen wird.

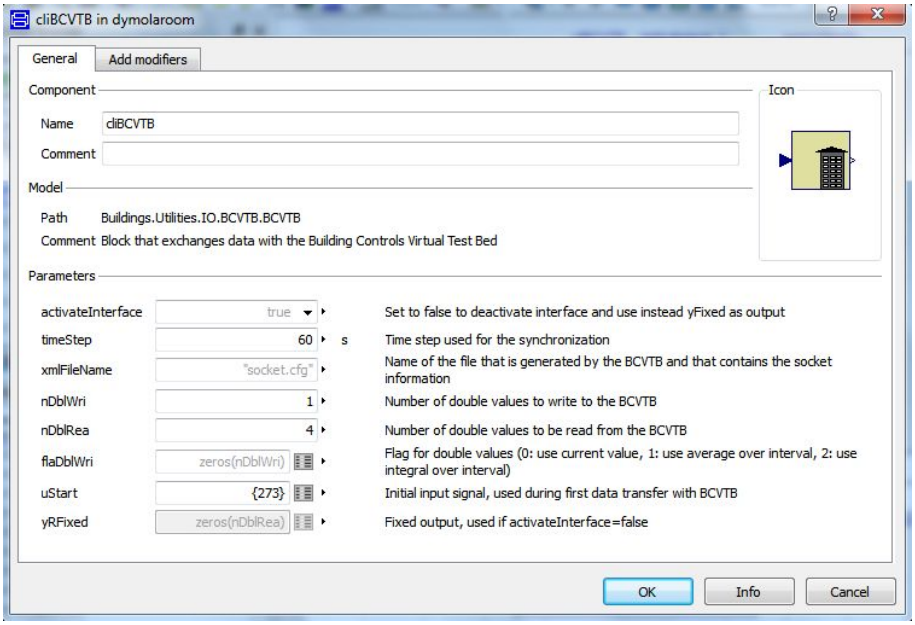

Abbildung 5.4: Optionen des BCVTB-Blocks für Dymola

Einige weitere wichtige Parameter, die im BCVTB-Block für Dymola gewählt werden können, sind *nDblWri* und *nDblRea*, welche festsetzen, wie viele Werte in jedem Zeitschritt dem BCVTB-Gesamtmodell übergeben werden bzw. von BCVTB erhalten werden sollen. *uStart* legt den Anfangsrückgabewert fest, der bei der ersten Synchronisation an BCVTB überge-

ben werden soll. Um am Teilmodell separate Simulationen, deren Notwendigkeit im Laufe der Modellentwicklung auftreten kann, durchführen zu können, ohne das Modell durch das Löschen des BCVTB-Blocks komplett vom Gesamtmodell abkoppeln zu müssen, kann der Parameter *activateInterface* auf *false* gesetzt werden und ein fixes Signal als Output des BCVTB-Blocks verwendet werden, welches in *yRFixed* eingestellt werden kann. Defaultmäßig besteht es aus einem Nullvektor mit *nDblRea* Einträgen, was der Größe desjenigen Vektors, der bei der mit BCVTB gekoppelten Ausführung des Modells von BCVTB in jedem Zeitschritt erhalten wird, entspricht.

5.1.2 Kommunikation mit MATLAB

Um den Datenaustausch mit MATLAB zu ermöglichen, wurde von den Entwicklern von BCVTB eine Funktion zur Verfügung gestellt, die die an BCVTB zu übergebenden Daten einliest und im Gegenzug die von BCVTB benötigten Werte als Rückgabewert liefert. Diese Funktion *exchangeDoublesWithSocket* muss in jedem Synchronisationsschritt des BCVTB Directors aufgerufen werden, was im Allgemeinen durch eine Schleife über alle Zeitschritte verwirklicht wird:

```
for i=1:(simTim/delTim+2)

    %%-------------------------------------------------------------------%%
    %%->>Code fuer die Aenderung des Rueckgabewerts in jedem Zeitschritt<<-%%
    %%-------------------------------------------------------------------%%
    [retVal, flaRea, simTimRea, dblValRea ] =
    exchangeDoublesWithSocket(sockfd, flaWri, length(u), simTimWri, dblValWri
        );
    simTimWri = simTimWri + delTim;
end
```

Zudem muss vor Beginn der ersten Kommunikation mit BCVTB ein BSD-Socket erstellt werden, was mittels

```
sockfd = establishClientSocket('socket.cfg');
```

geschieht. Nach dem letzten Datenaustausch wird der Socket durch den Aufruf von

```
closeIPC(sockfd);
```

wieder geschlossen.

5.1.3 Kommunikation mit Simulink und Simscape

Simulink- und Simscapemodelle, die in die Gesamtsimulation über BCVTB eingebunden werden sollen, werden ähnlich zu Dymolamodellen über Scripts aufgerufen, welche in diesem Fall natürlich als m-Files realisiert sind. Erneut dient diese Datei nur dazu, die Simulation des Modells mit den gewünschten Parametern aufzurufen.

```
load_system('simscapemotor');
sim('simscapemotor','MaxStep','60');
save_system('simscapemotor');
close_system('simscapemotor');
quit;
```

Der BCVTB-Block, der zur Kommunikation mit Simulink als auch Simscape dient, besteht im Wesentlichen aus den Blöcken, die den Code, der in MATLAB zum Austausch mit BCVTB dient, repräsentieren. Die genaue Aufgliederung des Blocks, wie er von den Entwicklern von BCVTB zur Verfügung gestellt wird, ist in Abb. 5.5 zu sehen.

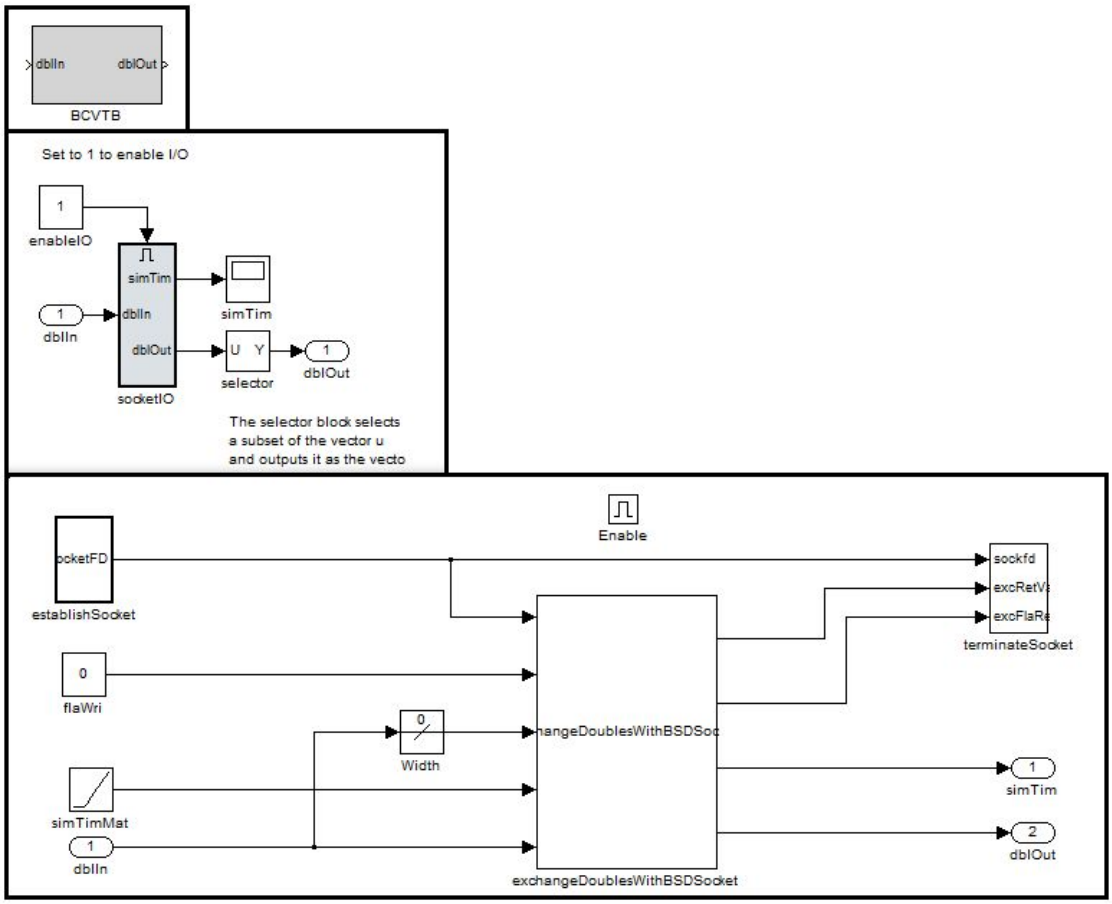

Abbildung 5.5: BCVTB-Block für Simulink/Simscape, der nach der Installation von BCVTB zur Verfügung steht (siehe [Wet12])

Im Gegensatz zum BCVTB-Block in Dymola öffnet der für den Austausch mit Simulink und Simscape konzipierte Block kein GUI mit Möglichkeiten zur Einstellung benötigter Parameter wie den Zeitschritten zur Synchronisation oder der Anzahl der Input- und Outputparameter. Der Aufbau des Blocks erlaubt eine problemlose Modifikation durch den User auf die Anforderungen des jeweiligen Beispiels mit der gewohnten Zugangsweise in Simulink. Andererseits kann der Zeitschritt, zu dem mit BCVTB ein Datenaustausch erfolgen soll, nicht so simpel wie im Fall von Dymola eingestellt werden.
Wird im übergeordneten BCVTB-Modell ein SDF Director verwendet, was eine fixe Schrittweite zur Kommunikation bedeutet, so wird zur Feuerung zum benötigten Zeitpunkt in Simulink kein Signal von BCVTB benötigt. In dieser Arbeit wurde die Aktivierung des Austauschs

so verwirklicht, dass die Simulationszeit in Simulink beobachtet wird und das Überschreiten eines Zeitpunkts modulo des Zeitschritts den BCVTB-Block in einem *If Action Subsystem* aktiviert (siehe Abb. 5.6).

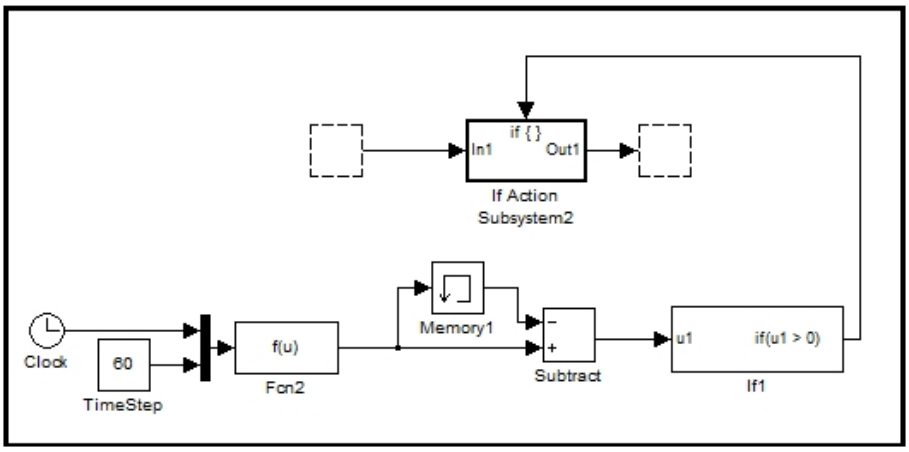

Abbildung 5.6: Aktivierung der Kommunikation von Simulink mit BCVTB zu fixen Zeitschritten

Im Fall der Verwendung eines Continuous Directors im umgebenden BCVTB-Modell muss ermittelt werden, wann BCVTB Daten von Simulink zum Austausch benötigt. Zu diesem Zweck werden diese Zeitpunkte zusätzlich vom BCVTB-Block an Simulink übergeben und mit der Simulationszeit von Simulink verglichen. Sofern diese innerhalb eines Toleranzintervalls um den vom Continuous Director vorgegebenen Synchronisationszeitpunkt liegt, wird das Subsystem durch das Senden eines diskreten Impulses aktiviert. Die soeben beschriebene Ermittlung des Zeitpunktes für die Kommunikation mit BCVTB bei Verwendung eines Continuous Directors ist in Abb. 5.7 veranschaulicht.

Da beide Arten der Aktivierung des Subsystems eine Iteration des genauen Synchronisationszeitpunktes benötigen, werden vom Solver in Simulink viele Schritte gesetzt, die im Nachhinein wieder verworfen werden müssen. Diese überflüssigen Zeitschritte wirken sich klarerweise auf die benötigte Rechenzeit aus, was wiederum die gesamte Simulation verlangsamt. Demnach hat diese Art der Kommunikation mit BCVTB zwar gegenüber dem zuvor beschriebenen Kommunikationsblock zwischen Dymola und BCVTB, der das exakte Erreichen des Synchronisationszeitpunktes ermöglicht, einen Nachteil, dieser könnte jedoch durch die Ermöglichung der Verwendung eines Continuous Directors mit variabler Schrittweite wiederum relativiert werden. Kapitel 6.1.1.4 wird jedoch zeigen, dass die Verwendung eines Continous Directors in der Praxis nur funktioniert, wenn die Simulation unter äußerst einschränkenden Bedingungen verläuft, welche die Verwendung eines Continuous Directors im Grunde wieder überflüssig machen.
Im Laufe dieser Arbeit konnte im Fall der Verwendung eines fixen Synchronisationszeitpunktes die Vermeidung dieser Iteration und somit der Zustandsberechnung zu allen redundanten Solverzeitschritten verwirklicht werden. Zum einen wurde dazu in den Parametern des

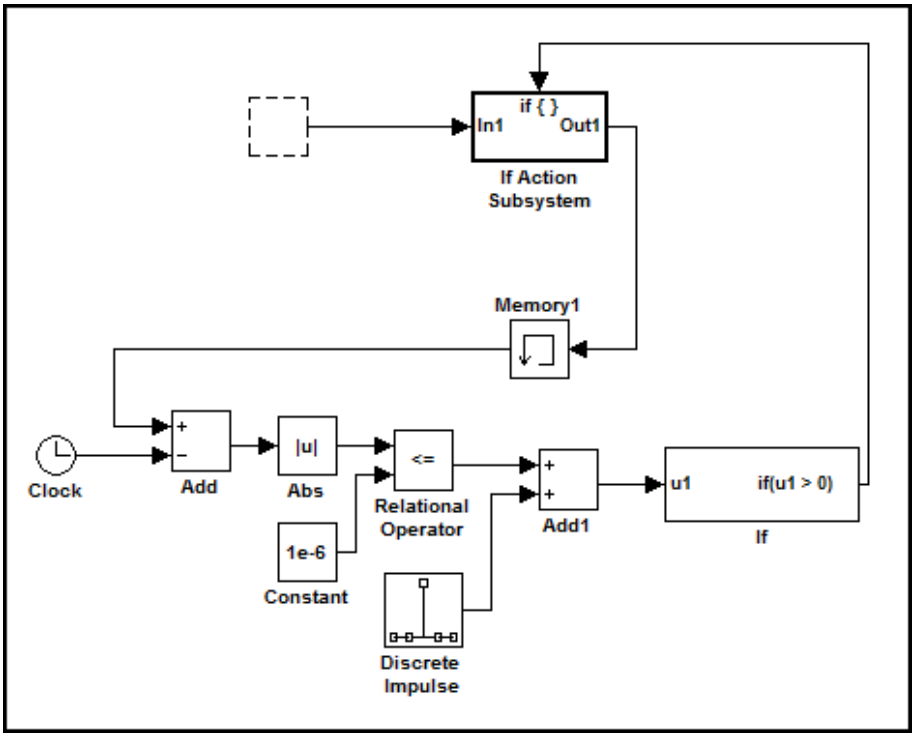

Abbildung 5.7: Aktivierung der Kommunikation von Simulink mit BCVTB zu variablen Zeitschritten

Blocks zur Ermittlung des Überschreitens der Synchronisationszeit (*If1*-Block in Abb. 5.6) die Option *Enable zero-crossing detection* deaktiviert. Um dennoch die Kommunikation zu jedem Synchronisationszeitpunkt zu gewährleisten, wurde zudem der Parameter *Sample time* im Block *TimeStep* auf den gewünschten, in diesem Fall fixen Zeitschritt zur Synchronisation gesetzt. Der Block, in dem die Sample Time gesetzt wird, kann beliebig gewählt werden. Der jeweilige in Simulink bzw. Simscape verwendete Solver ist gezwungen, zu jedem Vielfachen jeder im Modell auftretenden Sample Time einen Zeitschritt zu setzen, was das Feuern an BCVTB zum exakten Synchronisationszeitschritt ohne Iteration erlaubt. Ein Vergleich der gesetzten Zeitschritte in einem Simscapemodell mit bzw. ohne Definition einer Sample Time kann in Kapitel 6.1.1.3 verfolgt werden.
Da auch die Anzahl an Werten, die in jedem Zeitschritt an das BCVTB-Gesamtmodell zurückgegeben werden soll, nicht wie beim BCVTB-Block für Dymola eingestellt werden kann, werden die Input- bzw Outputsignale schlicht zu einem Vektor zusammengefasst, dessen Dimension je nach Modell sich der User ohne eigens dafür vorgesehene Parameter bewusst werden muss. Die Anzahl an Werten, die aus dem Rückgabevektor von BCVTB benötigt wird, kann in einem Selector innerhalb des BCVTB-Blocks für Simulink festgelegt werden (siehe zweites Kästchen in Abb. 5.5).

5.1.4 Kommunikation mit EnergyPlus

Da EnergyPlus ebenfalls an der University of California entwickelt wurde, sind seit Version 5.0 eigens für die Kommunikation mit BCVTB vorgesehene Objektoptionen inkludiert. Unter dem Punkt *External Interface*, der allgemein für die Anbindung an Drittanbieter-Software benötigt wird, muss für die Möglichkeit einer Co-Simulation ein *ExternalInterface*-Objekt erstellt werden, dessen einziger Parameter dessen Name ist, für den bis Version 7.0 ausschließlich *PtolemyServer* gewählt werden konnte (siehe Abb. 5.8). In der seit Oktober 2012 verfügbaren Version 7.2 wären auch die Werte *FunctionalMockupUnitImport* und *FunctionalMockupUnitExport* möglich; auf diese Optionen wird in dieser Arbeit jedoch nicht weiter eingegangen, da sie für die Co-Simulation mit BCVTB nicht relevant sind. Um Werte von BCVTB an EnergyPlus liefern zu können, muss zusätzlich mindestens ein Objekt der Klasse *ExternalInterface:Schedule*, *ExternalInterface:Variable* oder *ExternalInterface:Actuator* erstellt werden. Ein *ExternalInterface:Schedule*-Objekt kann wie ein gewöhnlicher EnergyPlus-Schedule verwendet werden. Der Schedule ähnelt *Schedule:Compact* und wird zu jedem Zeitschritt auf den Wert gesetzt, der von BCVTB erhalten wird. Als Parameter sind dessen Name, der Name des zu verwendenden *ScheduleTypeLimits*-Objekts sowie der Startwert anzugeben. Ein *ExternalInterface:Variable*-Objekt kann als Variable in *EnergyPlus Runtime Language*-Programmen verwendet werden. Dasselbe gilt für ein *ExternalInterface:Actuator*-Objekt, das generell einem *EnergyManagementSystem:Actuator* entspricht.

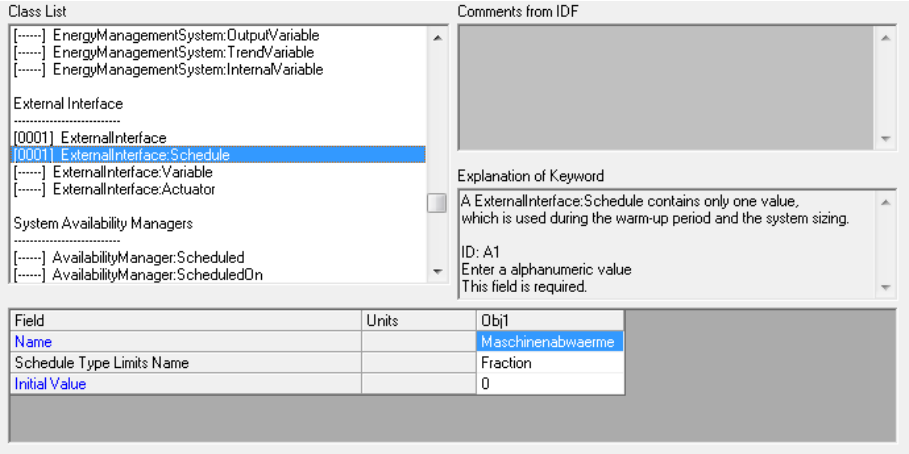

Abbildung 5.8: EnergyPlus-Objekte für die Co-Simulation mit BCVTB

Abgesehen von der Erstellung dieser Objekte im EnergyPlus-Modell muss für die Kommunikation mit BCVTB eine Konfigurationsdatei namens *variables.cfg* erstellt werden, die eine Beschreibung aller Variablen enthält, die von EnergyPlus zu BCVTB sowie von BCVTB zu EnergyPlus transferiert werden sollen. Listing 5.2 zeigt das Beispiel einer solchen Datei für die Übergabe eines Schedules von BCVTB an EnergyPlus und dessen Retournierung zweier Zonentemperaturen an BCVTB.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE BCVTB-variables SYSTEM "variables.dtd">
<BCVTB-variables>
<!-- Raumtemperatur in Zone 1 -->
  <variable source="EnergyPlus">
    <EnergyPlus name="ZONE␣1" type="Zone␣Mean␣Air␣Temperature"/>
  </variable>
<!-- Raumtemperatur in Zone 2 -->
  <variable source="EnergyPlus">
    <EnergyPlus name="ZONE␣2" type="Zone␣Mean␣Air␣Temperature"/>
  </variable>
<!-- Waermeabgabe an Zone 1 -->
  <variable source="Ptolemy">
    <EnergyPlus schedule="Maschinenabwaerme"/>
  </variable>
</BCVTB-variables>
```

Listing 5.2: XML-Datei zur Beschreibung der Variablen für den Austausch zwischen EnergyPlus und BCVTB

Weiters ist für eine Co-Simulation zu beachten, dass der Zeitschritt in EnergyPlus gleich dem Zeitschritt für die Gesamtsimulation in BCVTB gewählt werden muss, was unter anderem bedeutet, dass die Synchronisation nicht öfter als alle 60 Sekunden stattfinden kann, was die Co-Simulationsmöglichkeiten durchaus einschränkt und Ungenauigkeiten forciert.
Im Simulator Actor von BCVTB für den Zugriff auf EnergyPlus müssen sowohl der Name der idf-Datei als auch jener der verwendeten epw-Datei für die Wetterinformationen angegeben werden. Es ist nicht möglich, ein mit BCVTB gekoppeltes EnergyPlus-Modell ohne Wetterdatei zu simulieren. Falls konstante Wetterbedingungen gewünscht sind, muss eine epw-Datei erstellt werden, die eine solche Situation vorgibt. Als für den Benutzer wichtige Information ist zu bemerken, dass die Wetterdatei sich nicht wie die .idf-Datei in dem im Simulator Actor angegebenen Ordner befinden muss, sondern im Ordner *Weather Data*, der im Installationsordner von EnergyPlus zu finden ist.

5.2 Schrittweitendokumentation

Eines der interessantesten Ergebnisse einer Co-Simulation stellt die Analyse der unterschiedlichen Schrittweiten, die die einzelnen Solver getätigt haben, dar. Zu deren Dokumentation wurden die Zeitschritte, die in den jeweiligen Simulatoren auftreten, in Textdateien gespeichert.
Für MATLAB bzw. Simulink und Simscape konnte dies relativ gut in den Programmablauf integriert werden. Die Modifikationen im zugehörigen MATLAB Script als auch im Modell können in Listing 5.3 bzw. Abb. 5.9 nachverfolgt werden.

```
1 MYFILEPR=fopen('stepSimulink.txt','w');
2 load_system('simscapemotor');
3 set_param('simscapemotor/fhandle','Value',num2str(MYFILEPR));
4 sim('simscapemotor','MaxStep','60');
5 save_system('simscapemotor');
6 close_system('simscapemotor');
```

```
7 fclose(MYFILEPR);
8 quit;
```

Listing 5.3: Modifikation des MATLAB Scripts zur Schrittweitendokumentation

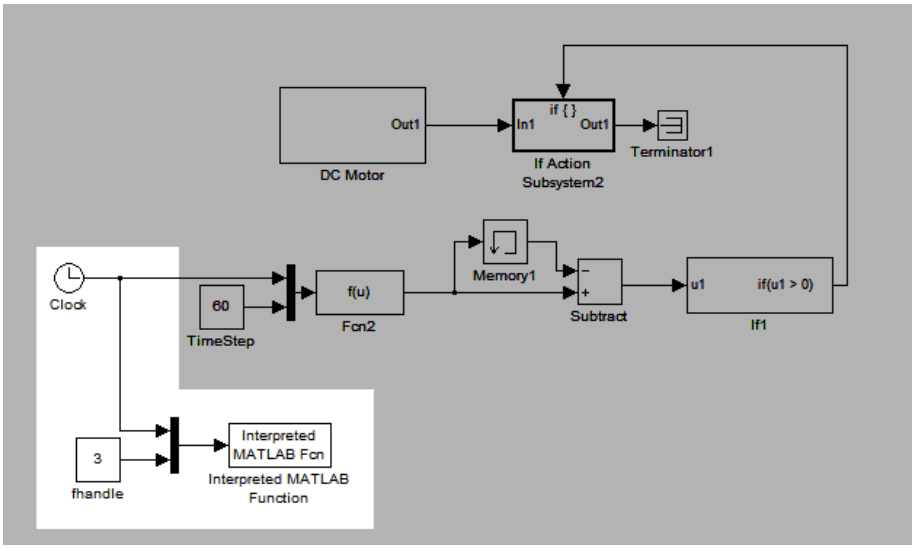

Abbildung 5.9: Modifikation des Simulinkmodells zur Dokumentation der Solverzeitschritte

Zunächst wird das .m-File zum Aufruf des .mdl-Modells dahingehend verändert, als eine Textdatei, die später die Zeitschritte beinhalten soll, erstellt und einem Filehandle zugewiesen wird (Zeile 1). Vor der Simulation des Simulink- oder Simscapemodells wird einer bereitgestellten Konstanten im Modell der Wert des Filehandles zugewiesen (Zeile 3 bzw. Block *fhandle*). Das Functionhandle sowie die aktuelle Simulationszeit werden zu einem Vektor zusammengefasst und in jedem Zeitschritt dem Funktionsblock *Interpreted MATLAB Function* übergeben, der die Funktion *printStep(u)* aufruft, welche lediglich dem Anhängen des neuen Zeitwerts - enthalten im ersten Eintrag des Signals- in eine neue Zeile der Textdatei, die durch den zweiten Eintrag des Vektors an die Funktion geliefert wird, dient:

```
function []=printStep(u)
fprintf(u(2),'%.16f\n',u(1));
end
```

Erst nach Simulationsende wird die Textdatei wieder geschlossen, was eine signifikante Verlängerung der Rechenzeit aufgrund eines unnötigen Öffnens und Schließens der Datei vermeidet.
Für die Dokumentation der Zeitschritte des von Dymola verwendeten Solvers wurde bislang keine Lösung gefunden, die das Öffnen und Schließen des Textfiles in jedem Zeitschritt umgeht. Durch das Einfügen der Codezeile

```
Modelica.Utilities.Streams.print(String(time), "stepDymola.txt");
```

in den Quellcode des Modells wird die Simulationszeit in jedem Zeitschritt an den Text der Datei angefügt, zudem wird jedoch die Datei zu jedem verwendeten Zeitpunkt geöffnet und wieder geschlossen. Dieser Prozess verzehnfacht die benötigte Rechenzeit pro Simulation, was nahelegt, die Solverschrittweiten nur gezielt für einige Experimente zu dokumentieren.

5.3 Raummodell, implementiert in Modelica via Dymola

5.3.1 Thermisches Festkörpercompartment

Das Raummodell in Dymola wurde als Compartmentmodell realisiert. Zu diesem Zweck wurde im Rahmen der Diplomarbeit von Matthias Rößler (siehe [Rö12]) ein Dymolablock entwickelt, der ein thermisches Compartment repräsentiert. Die Blockparameter sowie das zugrunde liegende Blockdiagramm, das mithilfe der von Dymola zur Verfügung gestellten graphischen Oberfläche modelliert wurde, werden in Abb. 5.10 gezeigt.

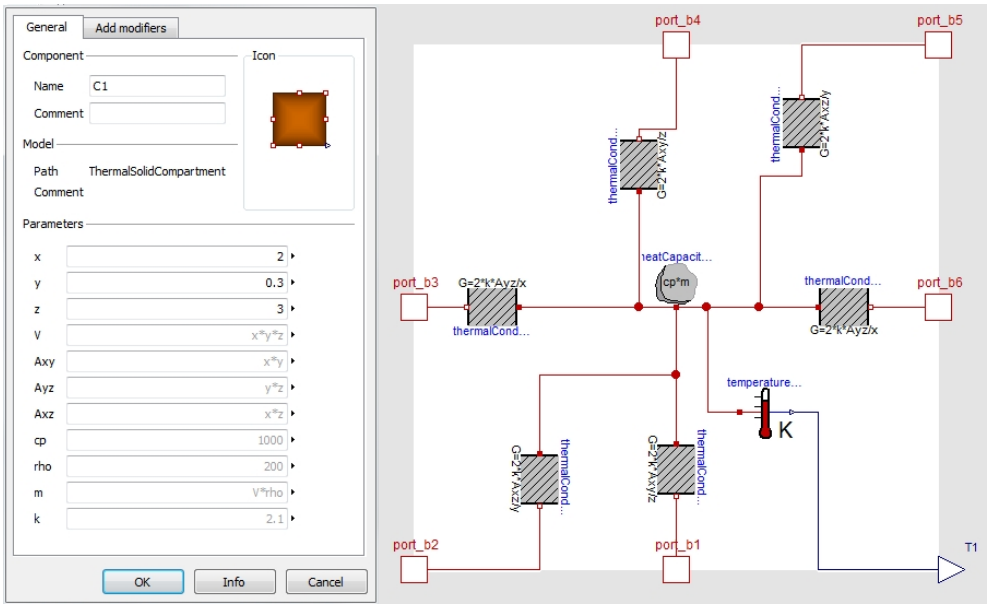

Abbildung 5.10: Im Rahmen von [Rö12] entwickeltes thermisches Compartment

Ein solches thermisches Compartment verkörpert einen Quader, dessen Abmessungen vom User als Parameter übergeben werden können. An jeder der Seitenflächen wird durch einen *Conductor*-Block Wärmeleitung ermöglicht. Der materialabhängige Wärmeleitkoeffizient *k* muss sowohl aufgrund der mit der Fläche höher werdenden Leitfähigkeit mit der Seitenfläche (z.B. Ayz) multipliziert als auch wegen der Abnahme durch den von der Seitenfläche bis zur Mitte des Compartments zurückgelegten Weges durch diesen (z.B. $x/2$) dividiert werden. Die zweite Hälfte des Weges wird von dem Konduktionsblock auf der gegenüberliegenden Seite berücksichtigt. Des Weiteren verfügt jedes thermische Compartment über eine gewisse Wärmekapazität, verwirklicht als *HeatCapacitor*-Block. Verbindun-

gen zu anderen Blöcken können über die sechs *heatPorts* an den Seitenflächen hergestellt werden.
Wie aus der obigen Beschreibung hervorgeht, wird Konvektion in diesem Modell in keiner Weise berücksichtigt. Dieser Ansatz erweist sich für gasförmige Stoffe natürlich als mangelhaft, da die Temperaturverteilung in der Luft hauptsächlich durch Teilchentransport geschieht. Aus diesem Grund wurde das oben beschriebene Compartment -unter der neuen Bezeichnung *ThermalSolidCompartment* - in dieser Arbeit nur für Raumelemente aus Feststoffen verwendet.

5.3.2 Thermisches Luftcompartment

Um dennoch das Raummodell in Dymola mit realen Systemen vergleichbar zu machen, wurde in dieser Arbeit versucht, das eben beschriebene Modell zu einem thermischen Compartment weiterzuentwickeln, das Konvektion berücksichtigt. Ein Teilchentransport als solcher ist mit einem Modellbildungsansatz, wie ihn Modelica verwendet, offensichtlich nicht implementierbar. Eine Möglichkeit, den so entstehenden Wärmetransport dennoch zu berücksichtigen, besteht durch die Verwendung des Konvektionsblocks von Modelica, dem der konvektive Wärmeübergangskoeffizient als Input übergeben wird. Neben der Erstellung eines Dymolablocks zur Berechnung von ebenjenem mussten zudem die Blöcke für die Wärmekapazität als auch die Konduktion neu implementiert werden.
Die hier auftretenden thermischen Prozesse werden aufgrund des fix vorgegebenen Volumens des Raumes als isochor angenommen. Demzufolge muss die Wärmekapazität C für den *HeatCapacitor*-Block mit der thermischen Masse m durch $C = c_v \cdot m$ anstatt, wie von Modelica vorgeschlagen, durch $C = c_p \cdot m$ berechnet werden, wobei c_v die isochore und c_p die isobare Wärmekapazität bezeichnet. Um die Temperaturabhängigkeit der Wärmekapazität berücksichtigen zu können, wurde in Anlehnung an den bereits existenten, von Modelica vorimplementierten *HeatCapacitor* der Block *AirHeatCapacitor* entwickelt. In diesem wird die isochore Wärmekapazität in Abhängigkeit der Compartmenttemperatur aus einem Polynom zweiten Grades berechnet, dessen Koeffizienten durch quadratische Regression von den aus [GC06] gewonnenen Daten in Tabelle 5.1 mithilfe des MATLAB-Befehls *polyfit* berechnet wurden.

Die thermische Masse eines Compartments wird im Laufe einer Simulation konstant gehalten, um ungewollten Wärmeverlust des Systems zu vermeiden. Zur Ermittlung der Masse wurde die Formel $m = V \cdot \rho$ herangezogen, wobei das Volumen V entsprechend den Abmessungen des thermischen Compartments als Parameter übergeben wird und die Dichte ρ für die übergebenen Anfangstemperatur aus einem ebenfalls aus Tabelle 5.1 gewonnenen Regressionspolynom berechnet und für weiteren Verlauf konstant auf diesem Wert gehalten wird. Das resultierende Fenster für die Blockparameter des *AirHeatCapacitor* kann in Abb.5.11 betrachtet werden.

Wie in Tabelle 5.1 zu sehen ist, ist die temperaturbedingte Änderung der Wärmeleitfähigkeit k von Luft ebenfalls nicht zu vernachlässigen. Aus diesem Grund wurde neben dem modelicaeigenen *HeatCapacitor* auch der *ThermalConductor*-Block adaptiert. Da der Wärmeleitkoeffizient in jedem Schritt intern von der an einem HeatPort abgegriffenen Temperatur des Compartments mithilfe des Regressionspolynoms für die aus Tabelle 5.1 gewonnenen

$T\ [K]$	$\rho\ [kg/m^3]$	$k\ [mW/(mK)]$	$\nu\ [10^{-7}m^2/s]$	Pr	$c_v\ [J/(kgK)]$
243.15	1.434	22.023	109.4	0.7161	718.68
253.15	1.3771	22.811	117.7	0.7143	718.58
263.15	1.3245	23.59	126.2	0.7126	718.68
273.15	1.2758	24.36	135	0.711	718.78
283.15	1.2306	25.121	144	0.7095	718.98
293.15	1.1885	25.873	153.2	0.7081	719.28
303.15	1.1492	26.618	162.6	0.7068	719.58
313.15	1.1124	27.354	172.3	0.7056	719.98
323.15	1.0779	28.082	182.2	0.7045	720.58
333.15	1.0455	28.804	192.2	0.7035	721.08
343.15	1.015	29.518	202.5	0.7026	721.78
353.15	0.9862	30.225	213	0.7018	722.58
363.15	0.959	30.925	223.7	0.7011	723.38
373.15	0.9333	31.62	234.6	0.7004	724.38
393.15	0.8858	32.989	257	0.6994	726.48

Tabelle 5.1: Temperaturabhängige Parameter für Luft bei einem Druck von 1 Bar, Daten aus [GC06]

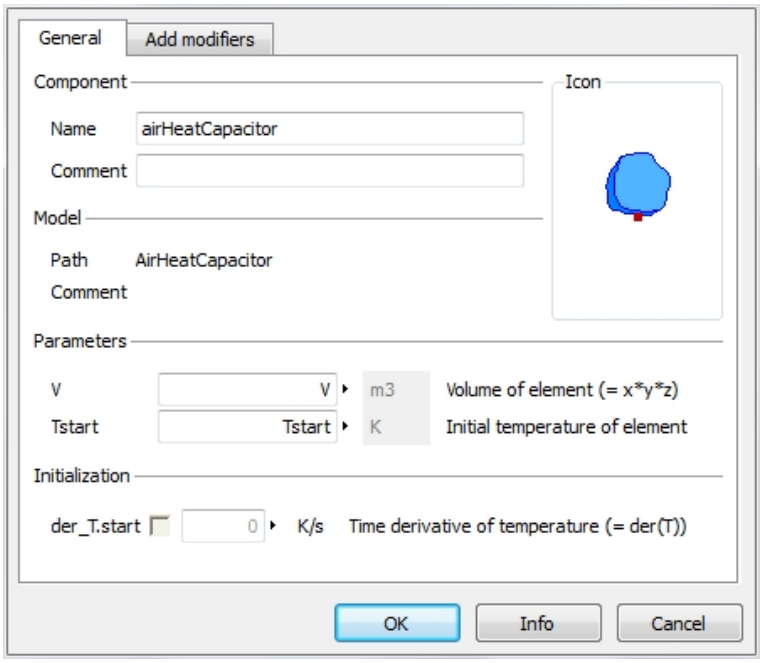

Abbildung 5.11: Parameter des in Modelica implementierten AirHeatCapacitor

Daten berechnet wird, bleibt als einziger Parameter der Weg (siehe Abb. 5.12), der sich aus dem Quotienten der Fläche und Dicke des Compartments bis zur Mitte zusammensetzt, wie bereits im vorhergehenden Kapitel für den Konduktionsblock im *ThermalSolidCompartment* erklärt wurde.

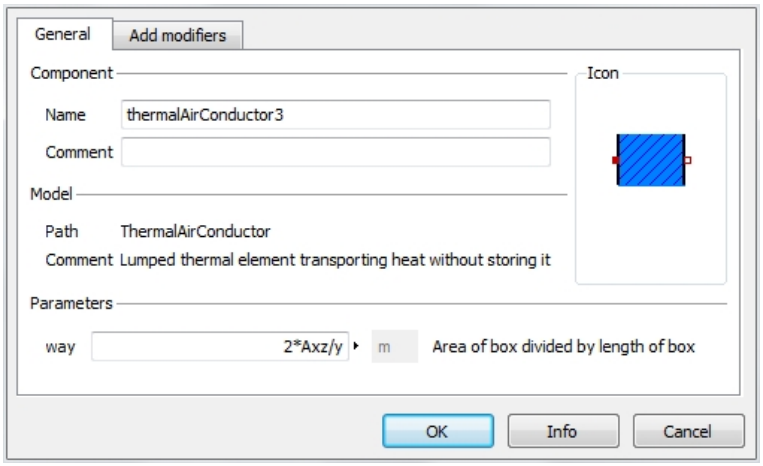

Abbildung 5.12: Parameter des in Modelica implementierten ThermalAirConductor

Die folgenden Daten sowie Formeln für die Ermittlung des konvektiven Wärmeübergangskoeffizienten wurden neben elementarphysikalischem Wissen aus [GC06] entnommen. Die Berechnung des konvektiven Wärmeübergangskoeffizienten α beruht zunächst auf Formel 5.1, wobei Nu für die Nusselt-Zahl, k für die Wärmeleitfähigkeit und l für die Anströmlänge steht.

$$\alpha = \frac{Nu \cdot k}{l} \tag{5.1}$$

Die Berechnung der Nusselt-Zahl sowie der Anströmlänge erfordert eine Unterscheidung der Position, an der die Konvektion stattfinden soll. Da ein thermisches Luftcompartment als Quader realisiert wird, muss zwischen Seitenflächen, oberer Fläche und unterer Fläche unterschieden werden. Da die Anströmlänge sich im Allgemeinen durch den Quotienten der Oberfläche des Körpers und des Umfangs seiner Projektionsfläche ergibt, erhält man für die Ober- und Unterseite des Quaders

$$l = \frac{x \cdot y}{2 \cdot (x + y)}, \tag{5.2}$$

wobei x und y die Breite bzw. Länge des Quaders angeben.
Für die Seitenflächen kann die Höhe der Flächen, welche gleichzeitig der Höhe z des Quaders entspricht, als Anströmlänge herangezogen werden, womit man

$$l = z \tag{5.3}$$

erhält.
Die Nusselt-Zahl für den Wärmeübergang auf der Oberseite des Quaders kann durch die empirische Gleichung für laminare Strömungen durch Formel 5.4 berechnet werden; die entsprechenden Gleichungen für die Unterseite und Seitenflächen zeigen Formel 5.5 bzw. 5.6.

$$Nu = 0.766 \cdot \left(Gr \cdot Pr \cdot \left(1 + \left(\frac{0.322}{Pr} \right)^{\frac{11}{20}} \right)^{-\frac{20}{11}} \right)^{\frac{1}{5}} \tag{5.4}$$

$$Nu = 0.6 \cdot \left(Gr \cdot Pr \cdot \left(1 + \left(\frac{0.492}{Pr} \right)^{\frac{9}{16}} \right)^{-\frac{16}{9}} \right)^{\frac{1}{5}} \tag{5.5}$$

$$Nu = \left(0.766 \cdot \left(Gr \cdot Pr \cdot \left(1 + \left(\frac{0.492}{Pr} \right)^{\frac{9}{16}} \right)^{-\frac{16}{9}} \right)^{\frac{1}{6}} \right)^{2} \tag{5.6}$$

Zur Auswertung dieser Ausdrücke wird zum einen die Prandtl-Zahl Pr sowie die Grashof-Zahl Gr benötigt. Letztere ist definiert als

$$Gr = \frac{g \cdot l^3}{\nu^2} \cdot \beta \cdot \Delta T \tag{5.7}$$

mit der Erdbeschleunigung g, der kinematischen Viskosität ν, dem räumlichen Wärmeausdehnungskoeffizienten β und der Temperaturdifferenz ΔT. Während der räumliche Wärmeausdehnungskoeffizient aufgrund der lediglich minimalen Änderung des Drucks mit dem Kehrwert der Temperatur in Kelvin angenähert werden kann, wurden für k, ν und Pr wie bereits zuvor für c_v und k durch die MATLAB-Funktion *polyfit* Regressionspolynome aus den Daten in Tabelle 5.1 ermittelt.
Durch die Verwendung der Formeln 5.1 bis 5.7 sowie der Regressionspolynome kann die Berechnung des konvektiven Wärmeübergangskoeffizienten für jede Fläche zu je einer Gleichung zusammengefasst werden, welche nur von der Temperatur im aktuellen Compartment und der Temperatur des jeweiligen an die betrachtete Fläche angrenzenden Compartments abhängt. Die Berechnung dieses Koeffizienten wird im *SI2SO*-Block *ConvCoeff* verwirklicht (siehe Abb. 5.13), wobei der User mit einem Drop-Down-Menü auswählen kann, ob der Koeffizient für Seitenflächen, Ober- oder Unterseite des Quaders berechnet werden soll. Des Weiteren sind die Anströmlänge sowie die umströmte Fläche anzugeben.

Die resultierende Blockstruktur des gesamten *ThermalAirCompartment* kann in Abb. 5.14 betrachtet werden.

Zusätzlich zu den modifizierten, jedoch vom Aufbau des *ThermalSolidCompartment* bekannten Blöcken wurde jeder *HeatPort* mit einem *ThermalConvection*-Block verbunden. Das Eingangssignal für diesen ist der konvektive Wärmeübergangskoeffizient k multipliziert mit der umströmten Fläche. Dieser Wert wird vom oben beschriebenen Block *ConvCoeff* als

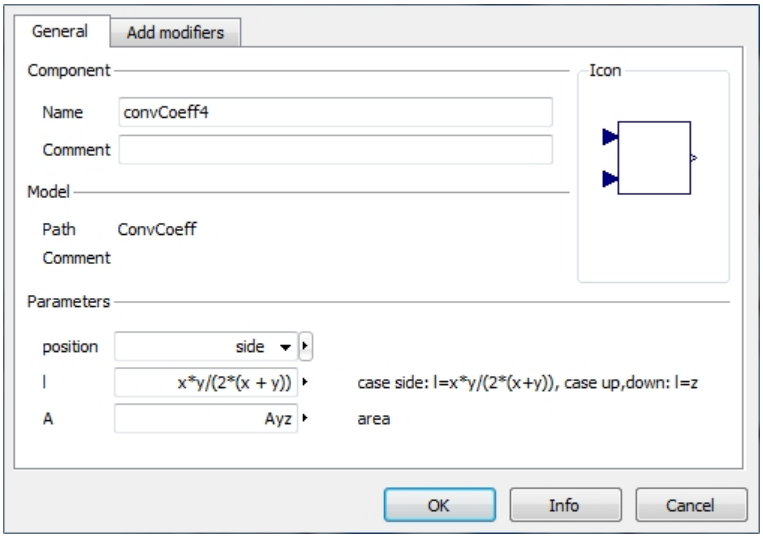

Abbildung 5.13: Parameter des in Modelica implementierten Blocks zur Berechnung des Konvektionskoeffizienten

Output geliefert. Als Eingang werden nur noch die an den entsprechenden *HeatPorts* abgegriffenen Temperaturen des Compartments sowie der betrachteten angrenzenden Fläche benötigt, da, wie zu Beginn dieses Abschnitts beschrieben, alle anderen zur Berechnung notwendigen Parameter durch Funktionen angenähert werden, die nur von diesen beiden Temperaturwerten abhängen.

Wie Abb. 5.15 zeigt, können die Wärmekapazität, die Dichte und der Wärmeübergangskoeffizient im Gegensatz zum Festkörpercompartment nicht mehr als Parameter angegeben werden, da sie intern berechnet werden. Zusätzlich zu den räumlichen Abmessungen des Compartments kann jedoch die Anfangstemperatur im Compartment gewählt werden.
Abhängig von der Raumgröße und der gewünschten Unterteilung werden für die resultierenden Raummodelle in 6.1 unterschiedlich viele dieser thermischen Compartments mit den jeweiligen Abmessungen über die entsprechenden *heatPorts* zusammengefügt.

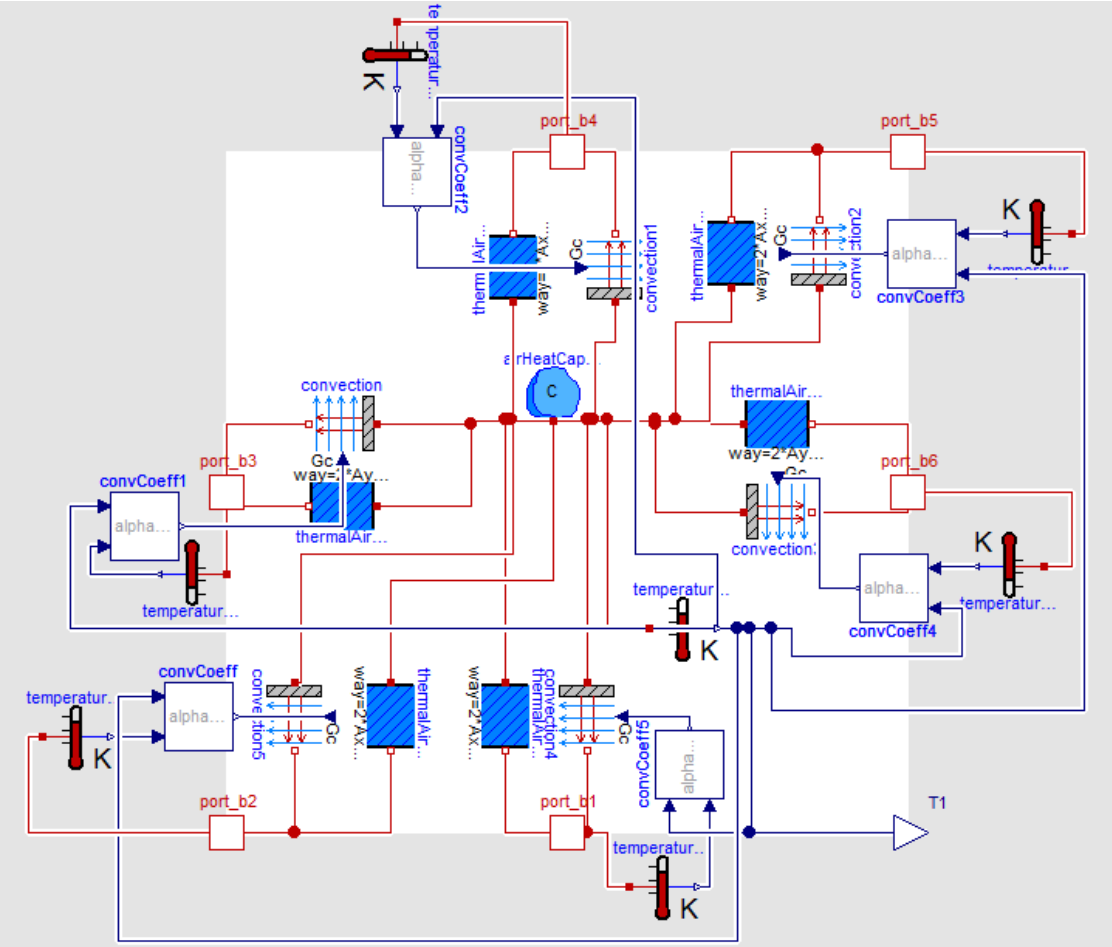

Abbildung 5.14: Blockdiagramm des in Modelica implementierten thermischen Luftcompartments

5.4 Einfaches Maschinenmodell, implementiert in Simscape

Eine der Maschinen, die in das Modell der Produktionshalle integriert werden sollen, wurde als Gleichstrommotor in Simscape modelliert. Das in Abb. 5.16 gezeigte Blockdiagramm besteht aus drei Kreisläufen, welche verschiedenen physikalischen Domänen entsprechen. Dem elektrischen Kreislauf wird die anliegende Spannung mittels einer *Controlled Voltage Source* vorgeschrieben. Durch die Vorgabe mithilfe eines *Pulse Generators* wird nur zu den gewünschten Arbeitsstunden der Maschine Spannung angelegt. Die Verbindung zwischen dem elektrischen und dem mechanischen Kreislauf stellt der *Rotational Electromechanical Converter* dar, der die resultierende elektrische Energie in Rotationsenergie umwandelt. Die von der Maschine an die Umgebung abgegebene Wärme muss mithilfe eines Funktionsblocks vom Spannungsabfall am elektrischen Widerstand berechnet werden. Die durch $P = U \cdot I = U^2/R$ in dieser Funktion berechnete Leistung wird dem thermischen Kreislauf

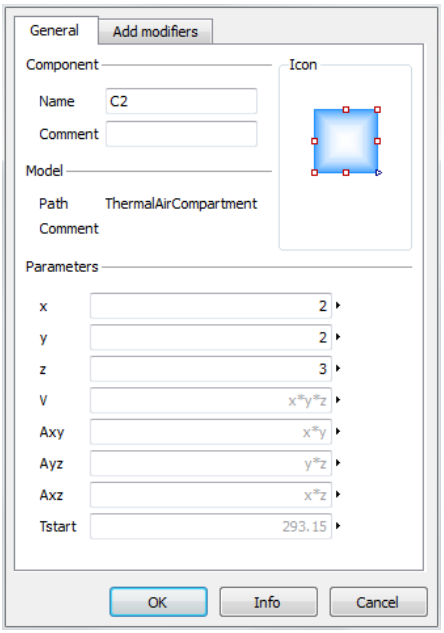

Abbildung 5.15: Parameter des in Modelica implementierten thermischen Luftcompartments

über eine Wärmequelle (*Ideal Heat Flow Source*) zugeführt. Der resultierende Wärmefluss von der Maschine an die Produktionshalle wird schließlich als Output zurückgeliefert.

5.5 Detailliertes Werkzeugmaschinenmodell, implementiert in Simscape

Da die Möglichkeiten der Co-Simulation mit BCVTB auch im Hinblick auf die Komplexität der Einzelmodelle erwogen werden sollen, sei in diesem Abschnitt das Modell eines Antriebsstranges der Hauptspindel einer realen Werkzeugmaschine, die im Zuge von [Hei12] entwickelt wurde, kurz erklärt. Abbildung 5.17 zeigt einen Teil des Modells, aus dem die Abwärme des Asynchronmotors als auch die durch mechanische Reibung entstehende Wärme in späterer Folge an das Modell der Halle übergeben werden sollen.

Das Modell des Asynchronmotors, das im großen, grauen Subsystem namens *AsynchronousMachine* untergebracht wurde, wird durch den Input dreiphasiger Wechselspannung betrieben, was den ersten Unterschied zu den anderen in dieser Arbeit verwendeten Motoren, die mit Gleichspannung betrieben werden, darstellt. Allein für die Simulation dieses Motors ist ein weitaus komplexeres System zu lösen, als es in den anderen Maschinenmodellen der Fall ist. Durch den Asynchronmotor wird über einen Zahnriementrieb (zu sehen im etwas kleineren, grau eingefärbten Subsystem namens *GearBeltDrive*) die Spindel bewegt, die in weiterer Folge eine Drehung des Werkstücks bewirkt. Die Abwärme des Motors sowie die durch die Drehung bedingte mechanische Reibung werden einem thermischen Kreislauf zugeführt, durch den die an die Umgebung abgegebene Wärme ermittelt wird.

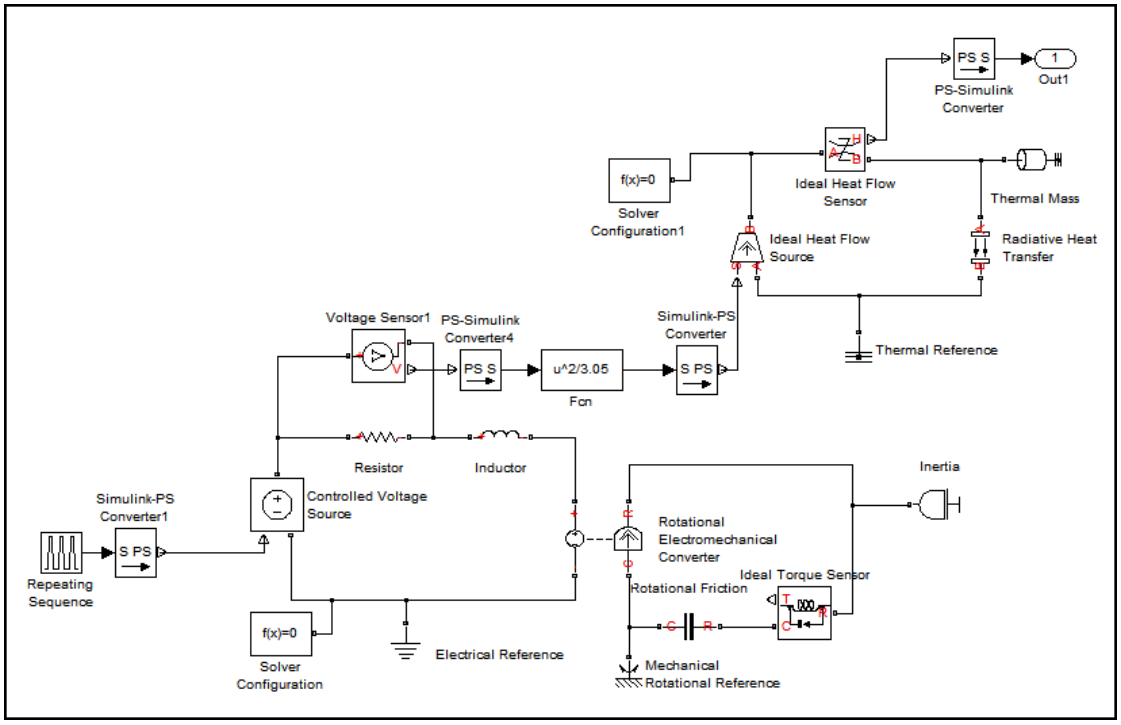

Abbildung 5.16: Modell eines Gleichstrommotors, implementiert in Simscape

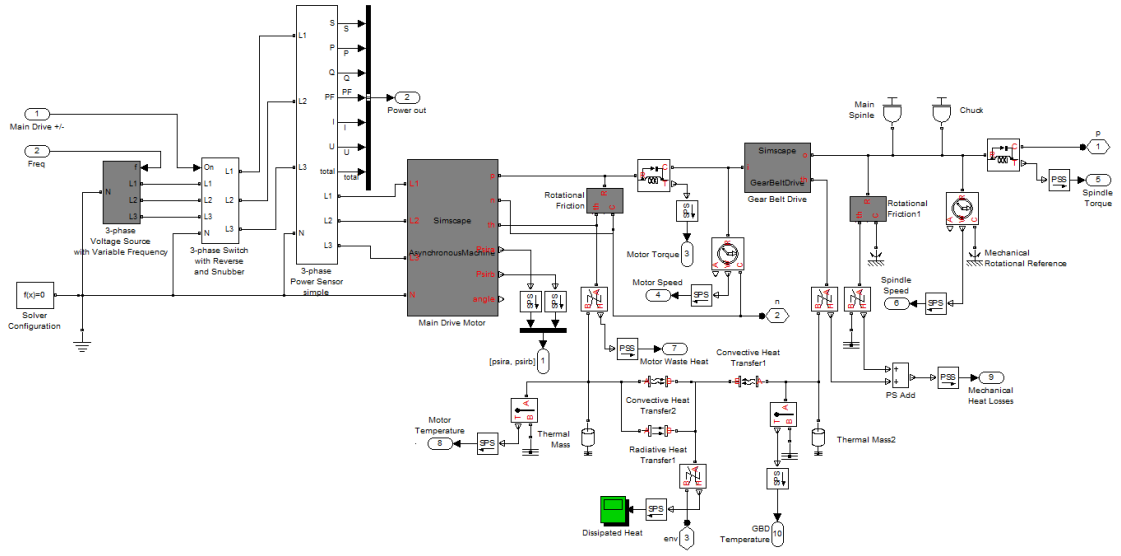

Abbildung 5.17: Modell des Antriebsstrangs einer Werkzeugmaschine, implementiert in Simscape, entnommen aus [Hei12]

5.6 Maschinenmodell, implementiert in Modelica via Dymola

Zum Vergleich der einzelnen Modellbildungs- und Simulationsumgebungen wurde mithilfe von Dymola ebenfalls ein Gleichstrommotor modelliert. Die Komponenten des Dymolamodells entsprechen im Wesentlichen jenen, die auch in Simscape herangezogen wurden, allerdings werden in Dymola deutlich weniger Blöcke zur Realisierung derselben benötigt, siehe Abb. 5.18. Dymola Version 2012 FD01 verwendet den Modelica Standard 3.2, in dem alle dissipativen Elemente über optionale HeatPorts verfügen, was bedeutet, dass die abgegebene Wärme direkt an diesem Port abgegriffen und ohne zusätzlichen Kalkulationsaufwand weiterverwendet werden kann.

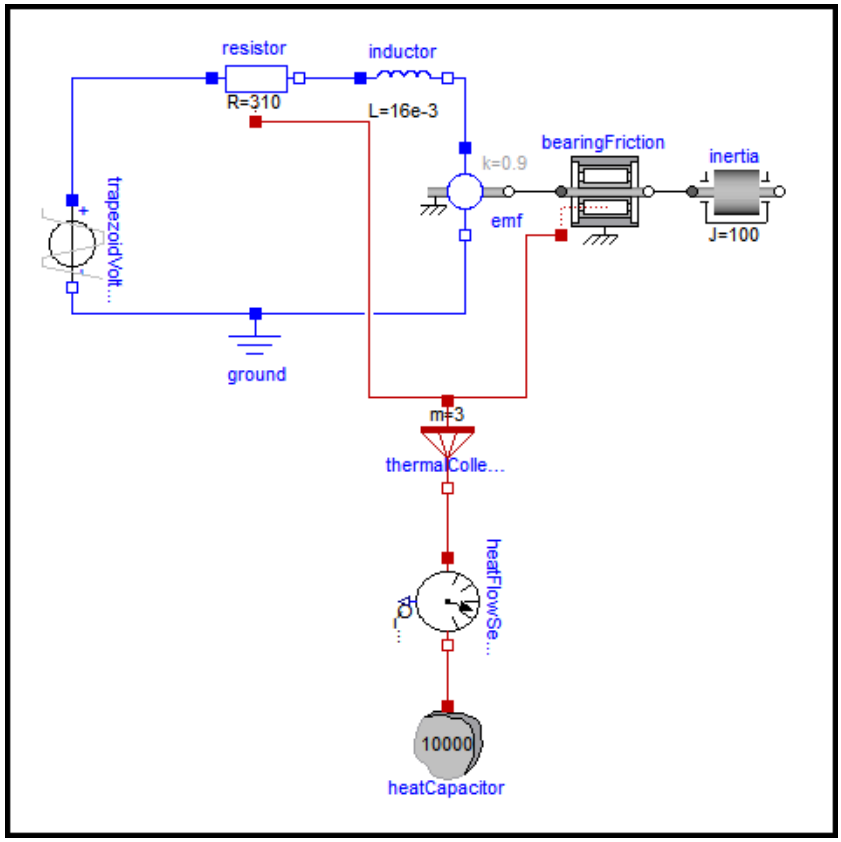

Abbildung 5.18: Modell eines Gleichstrommotors, implementiert in Modelica via Dymola

5.7 MATLAB-Datenmodell der abgegebenen Wärme einer realen Maschine

Wie bereits in Kapitel 1.1 erwähnt, entstand die Idee zur thermischen Co-Simulation von Maschinen in einer Produktionshalle in Anlehnung an eine bereits existente Fabrik, deren

Energiehaushalt simuliert werden soll. Die Wärmeemission einer dieser realen Maschinen wurde über die Dauer von einem Tag stündlich gemessen und in einer Excel-Datei dokumentiert. Um diese Daten an die Halle zu transferieren, wurde eine MATLAB-Funktion herangezogen. Ausschnitte aus dem zugrundeliegenden MATLAB-Code zeigt Listing 5.4.

```
1  % Initialize variables
2  retVal    = 0;  %return value
3  flaWri    = 0;  %
4  flaRea    = 0;  %
5  simTimWri = 0;  % simulation time to write
6  simTimRea = 0;  % simulation time to read
7  u=0;
8
9  simOut = xlsread('Waerme.xlsx');
10 simOut(:,1)=simOut(:,1)*3600;
11 simOut(:,2:3)=simOut(:,2:3)*10;
12
13 simTim=7*24*3600;
14 delTim=60;
15
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 % Establish the socket connection
18 sockfd = establishClientSocket('socket.cfg');
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 % Exchange data (call this at each time step)
21 % ... (loop over each time step)
22 for i=0:(simTim/delTim+1)
23     x=simOut(floor(mod(simTimRea,24*3600)/3600)+1,2); %Leistung
           Bearbeitungszentrum
24
25     [retVal, flaRea, simTimRea, u ] = ...
26     exchangeDoublesWithSocket(sockfd, flaWri, length(u), simTimWri, ...
27     x);
28
29     simTimWri = simTimWri + delTim;
30 end
31
32 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
33 % Close socket at the end of the simulation
34 closeIPC(sockfd);
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 % Exit MATLAB
37 quit
```

Listing 5.4: MATLAB Code zum Auslesen und Weitertransfer der gemessenen Wärmeemission einer Schneidemaschine

Die Zeilen 1-7 beinhalten lediglich die Initialisierung der für den späteren Austausch notwendigen Variablen. In der Variable *simOut* werden die durch den Befehl *xlsread* aus dem Excel-File ausgelesenen Daten gespeichert. Die Werte in der ersten Spalte entsprechen den Messzeitpunkten, die in Zeile 10 in Sekunden umgewandelt werden. Die Wärmeleistung wird in Zeile 11 in Watt konvertiert. In der Schleife über jeden Zeitschritt (Zeilen 22

bis 30) wird überprüft, in welcher Stunde eines Tages sich der aktuelle Simulationszeitpunkt befindet (Zeile 23), um die zuletzt gemessene Wärmeemission der Maschine - unter der Annahme, dass diese sich täglich periodisch wiederholt - in weiterer Folge mithilfe der bereits in Kapitel 5.1.2 erklärten Funktion an BCVTB zu liefern (Zeilen 25-27).

5.8 Temperaturregelung in Simulink

Um die Temperatur in der Maschinenhalle in einem für etwaige menschliche Arbeiter angenehmen Rahmen zu halten, wurde eine Dreipunktregelung mit Hysterese in Simulink implementiert. Das Blockdiagramm dazu ist in Abb. 5.19 zu sehen.

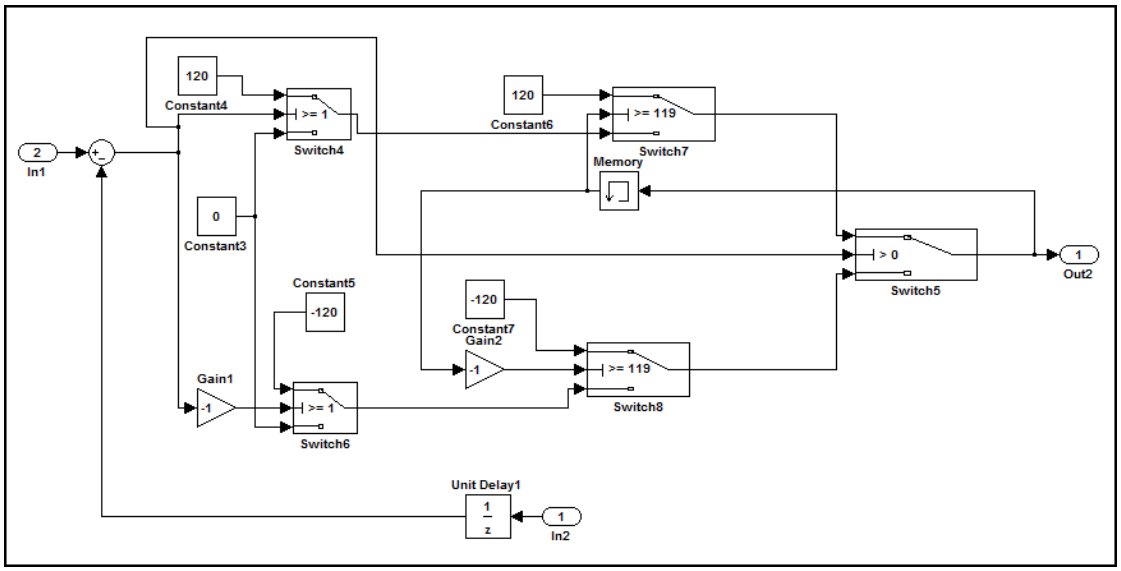

Abbildung 5.19: Dreipunkt-Temperaturregelung mit Hysterese, implementiert in Simulink

Dem System wird sowohl die aktuelle Raumtemperatur, die meist in einem der Compartments gemessen wird, übergeben, als auch die gewünschte Temperatur. Sobald der Betrag der in jedem Zeitschritt gebildeten Differenz dieser beiden Werte 1 Kelvin überschreitet, beginnt die Regelung zu kühlen bzw. zu heizen, indem positive bzw. negative Wärmeleistung (im in Abb. 5.19 gezeigten Fall von 120 Watt) zurückgeliefert wird. Sobald die gemessene Raumtemperatur wieder unter bzw. über dem Sollwert liegt, stoppt das System die Kühlung bzw. Heizung. Zu diesem Zweck ist es notwendig, zu wissen, ob das System im vorhergehenden Zeitschritt gekühlt oder geheizt hat, was mittels *Memory*-Block realisiert wird. Sobald die Raumtemperatur von Neuem um mindestens 1 Kelvin von der Solltemperatur abweicht, beginnt das System erneut zu kühlen oder zu heizen.

KAPITEL 6

Simulation und Resultate

6.1 Fallstudien

Der Beginn dieses Abschnitts erlaubt die Validierung der in Kapitel 5 entwickelten Komponenten für die Modellierung eines Raummodelles in Dymola. Zur Evaluierung der Möglichkeiten und Resultate der Co-Simulation mit BCVTB werden im Folgenden die Vor- und Nachteile einer Kopplung von in der gleichen Umgebung erstellten Modellen über eine Co-Simulation im Gegensatz zur softwareinternen Kopplung erörtert. Weitere Experimente behandeln die Kopplung von mehreren Instanzen aller verwendeten Simulatoren für die Simulation eines gesamten Maschinenhallenmodells sowie den Vergleich unterschiedlicher Modellierungsarten für dasselbe Modellproblem. Die Dokumentation der von den einzelnen Solvern getätigten Schrittweiten in jeder Simulation veranschaulicht die Methode des Datenaustauschs und die Plausibilität der Verwendung einer Co-Simulation.

6.1.1 Simulationen unter Verwendung des in Modelica via Dymola implementierten Raummodells

6.1.1.1 Validierung des Dymola-Raummodells anhand eines Modellversuchs

Um das Modell der thermischen Compartments, die für die Erstellung der Dymola-Raummodelle verwendet werden, zu validieren, wurde bereits im Zuge der Diplomarbeit von Matthias Rößler (siehe [Rö12]) ein Miniaturraum aus Styropor mit den Abmessungen $53.5cm \times 34cm \times 24cm$ gebaut. Eine Abbildung davon kann in Abb. 6.1 betrachtet werden.

Der Raum kann gedanklich in acht quaderförmige Compartments mit den Abmessungen $26.75cm \times 17cm \times 12cm$ unterteilt werden, von denen jedes in weiterer Folge in Dymola durch einen *ThermalAirCompartment*-Block realisiert wurde. Zur Simulation einer Wärmequelle wurde in einem der Miniaturcompartments im Styroporraum ein Lötkolben angebracht. Zur Bestimmung des resultierenden Temperaturverlaufs im Raum wurden zudem in jedes Compartment durch kleine Löcher in den Seitenwänden Sensoren so eingebracht,

Abbildung 6.1: Miniaturraummodell aus Styropor

dass die Temperatur in der Mitte jedes Compartments gemessen werden konnte. Abbildung 6.2 zeigt das zur Validierung erstellte Modell des Styroporraumes in Dymola.

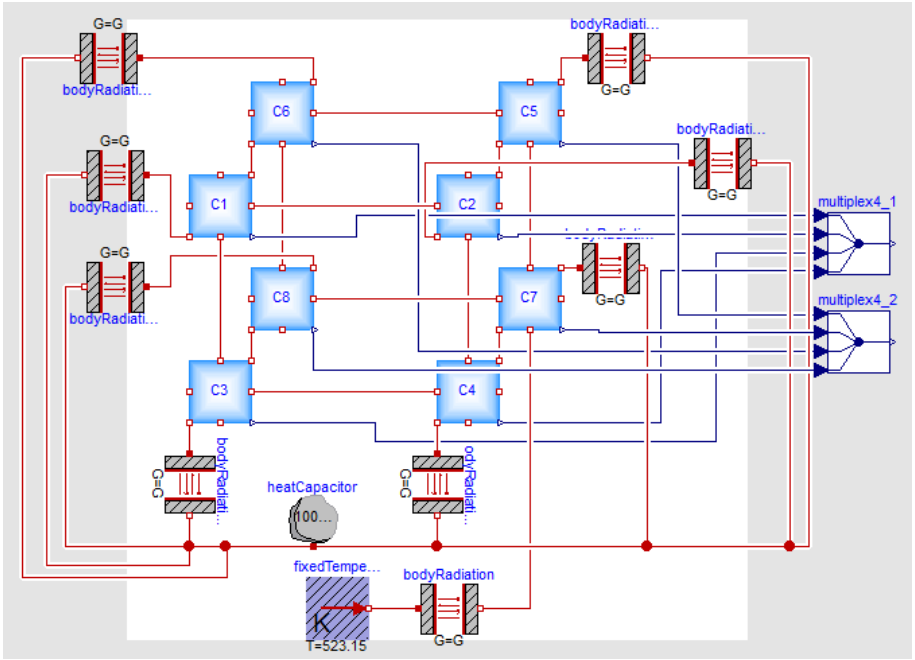

Abbildung 6.2: Miniaturraummodell, implementiert in Dymola

Da für die Einbringung der Temperatursensoren kleine Löcher notwendig waren, kann nicht von einem komplett isolierten Raum ausgegangen werden. Demnach wird der *HeatPort* einer Außenwand jedes Raumes für eine im *BodyRadiation*-Block angegebene Fläche von $9mm^2$ mit einem als Verkörperung der Umgebung sehr hoch zu parametrisierenden

HeatCapacitor verbunden. Die Temperatur des Lötkolbens, für den eine Oberfläche von $40mm^2$ angenommen wird, wird dem Compartment 7 über einen *FixedTemperature*-Block zugeführt. Die über 50 Minuten gemessene Temperatur in den Compartments des Styropor-Miniaturmodells ist in Abb. 6.3 zu sehen.

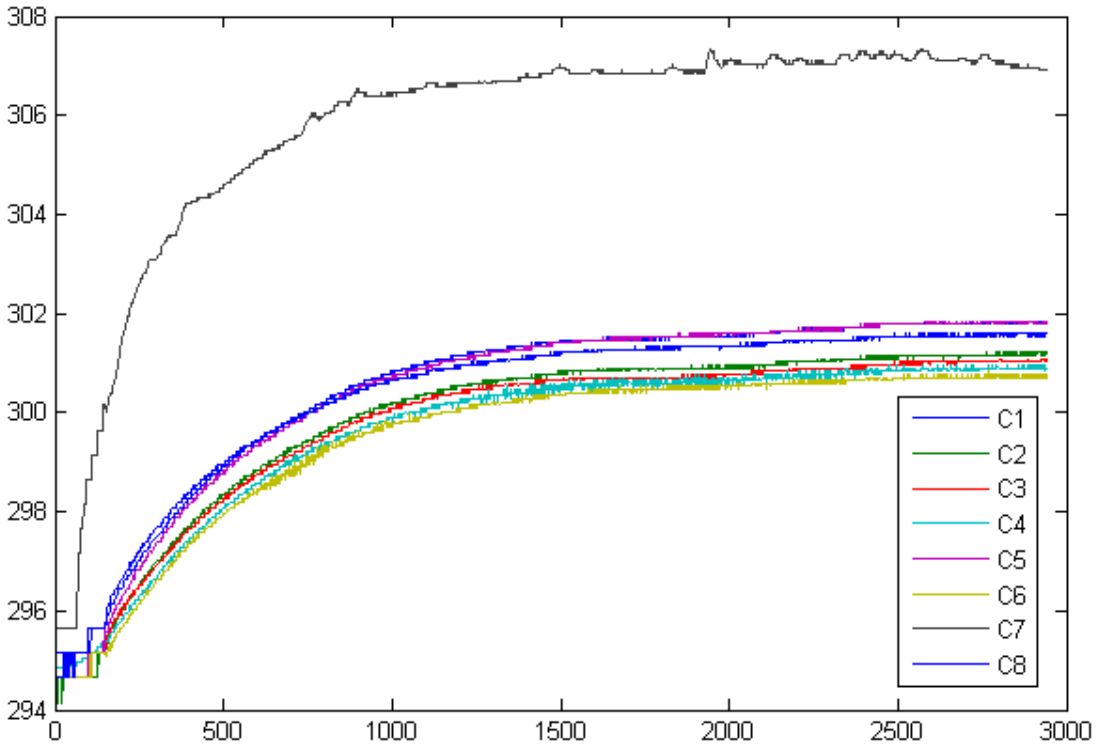

Abbildung 6.3: Über 50 Minuten gemessene Temperatur in den Compartments des Styropormodells

Abb. 6.4 zeigt das Resultat der Simulation des Dymola-Miniaturraummodells über 50 Minuten.

Obwohl der Temperaturverlauf zunächst in der Simulation vollkommen anders als zum realen Experiment erscheinen mag, ist zu beachten, dass die nach 50 Minuten erreichte Temperatur in den realen Compartments vor allem in den Compartments, in denen sich der Lötkolben nicht befindet, in der Simulation weitgehend erreicht wird. Der geringere Unterschied zwischen dem Compartment mit der Wärmequelle zu den übrigen Compartments kann dadurch erklärt werden, dass diese Compartments in der Computersimulation schneller auf die Temperaturerhöhung im Compartment 6 reagieren als im realen Experiment.
Die unterschiedliche Form der Kurven lässt sich dadurch erklären, dass die warme Luft im Experiment immer wieder aufsteigt um nach einiger Zeit wieder abzusinken, wodurch eine schnelle Vermischung der gesamten Raumluft veranlasst wird. Diese führt dazu, dass der Temperaturverlauf im realen Experiment in Stufen, die einer Dauer von etwa 50 Minu-

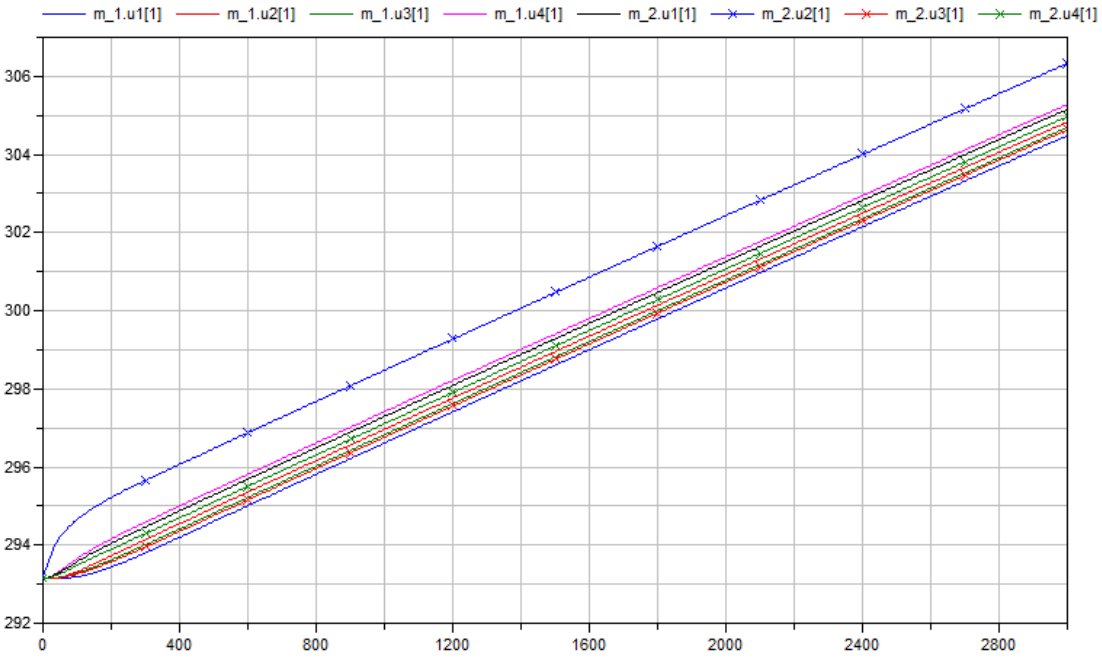

Abbildung 6.4: Temperaturverlauf in den Compartments als Simulationsergebnis des Dymolamodells über 50 Minuten

ten entsprechen und der Form von Abb. 6.3 ähneln, ansteigt, bis zu einem real nicht mehr simulierten Zeitpunkt in allen Compartments die Temperatur des Lötkolbens erreicht bzw. infinitesimal genau angenähert wird. Diese Stufen treten im Dymolamodell wegen dieser nicht verwirklichbaren Bewegung nicht auf; für die Simulation des gleichen Modells über einen sehr großen Zeitraum ähnelt die Form der Kurve jener einer Stufe aus dem realen Experiment. Abb. 6.5 zeigt den resultierenden Temperaturverlauf der Simulation über zwei Tage.

Zudem stimmen die Werte nach 50 Minuten in beiden Experimenten annähernd überein, was insgesamt darauf schließen lässt, dass die Kurve aus der Computersimulation lediglich einer Glättung der gemessenen Daten entspricht. Insgesamt kann das in Modelica implementierte Modell des Raumes als zulässig befunden werden.

6.1.1.2 Kopplung des Raummodells mit einer in Modelica via Dymola implementierten Maschine

Das erste Maschinenmodell, das in mit dem Raummodell gekoppelt und simuliert werden soll, ist das in Kapitel 5.6 beschriebene Dymola-Maschinenmodell. Als Parameter für die Simulation dieses Motormodells wurden $R = 310\Omega$, $L = 16 \cdot 10^{-3}H$ und $k = 0.9Nm/A$ gewählt.
Die gekoppelte Simulation kann entweder über das Co-Simulationstool BCVTB geschehen, da jedoch beide Modelle mithilfe von Dymola modelliert wurden, kann sie auch direkt in

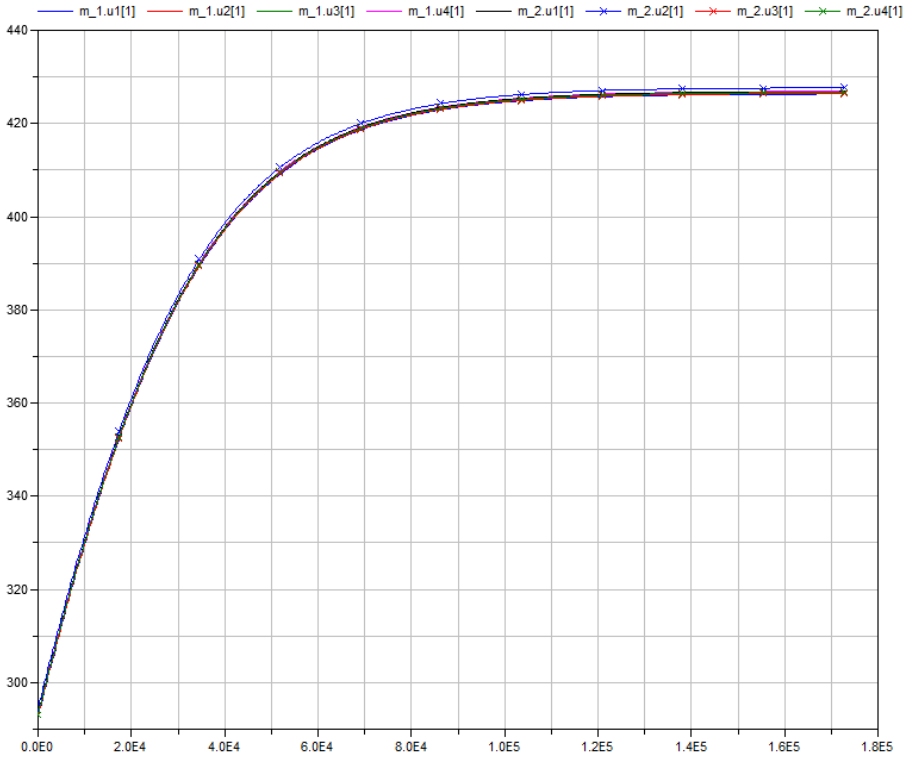

Abbildung 6.5: Temperaturverlauf in den Compartments als Simulationsergebnis des Dymolamodells über zwei Tage

Dymola erfolgen. Die Vor- und Nachteile beider Kopplungsmöglichkeiten werden in diesem Abschnitt erörtert.

Dymolainterne Kopplung

Eine Übersicht der Methode zur Kopplung in einem Dymolamodell ist in Abbildung 6.6 zu sehen.

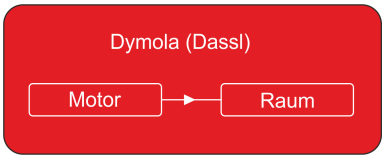

Abbildung 6.6: Veranschaulichung der Simulation des Raummodells und des Maschinenmodells über eine Dymolainstanz

Für die Modellierung des Raumes wurden 6 thermische Luftcompartments mit den Abmessungen $2m \times 2m \times 3m$ verbunden. Um ein Fenster, dessen Scheibe die Außentemperatur von 10^oC annimmt, zu simulieren, wurde mit einem äußeren *HeatPort* von Com-

partment 3 eine Temperaturquelle über eine Fensterfläche von $2m^2$ verbunden. Ansonsten wird der Raum nach außen hin als isoliert angenommen, weshalb an den äußeren *HeatPorts* der Compartments keine weiteren Verbindungen, etwa zu Festkörpercompartments oder *HeatCapacitors*, welche die Umgebung repräsentieren könnten, zu sehen sind. Um das Modell der Maschine direkt in Dymola in das Raummodell zu integrieren, wurde dem in in Kapitel 5.6 beschriebenen Maschinenmodell ein Output-Port an den Wärmeflusssensor hinzugefügt, sodass das Modell einfach im Raummodell geladen und als eigener Block hinzugeschaltet werden kann. Der Ausgang des Maschinenmodell-Blocks, welcher der abgegebenen Wärme entspricht, wird über einen *PrescribedHeatFlow*-Block einem der Compartments im Raum übergeben, wie in Abb. 6.7 zu sehen ist.

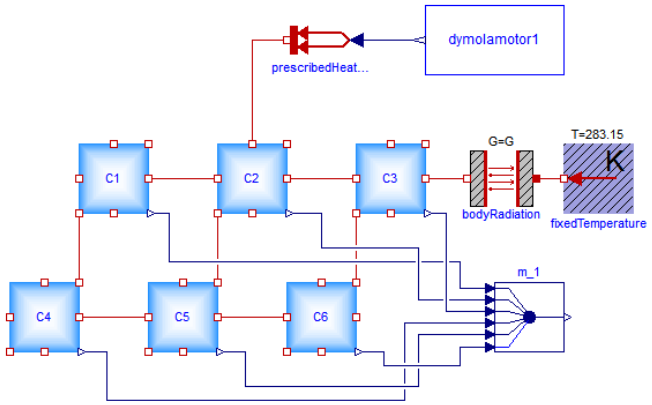

Abbildung 6.7: Modell eines Raumes mit einer Gleichstrommaschine, implementiert in Modelica via Dymola

Um acht Uhr morgens wird begonnen, an den Gleichstrommotor eine Spannung anzulegen, die linear ansteigt und nach zehn Minuten ihren Betriebswert von 230 Volt erreicht. Um 16 Uhr wird die Maschine in einer Dauer von fünf Minuten heruntergefahren, was erneut durch einen linearen Spannungsabfall bis zum Erreichen von 0 Volt realisiert wird. Der resultierende Wärmefluss vom Motor an die Umgebung für die Simulation des Modells über einen Werktag kann in Abb. 6.8 betrachtet werden.

Der daraus resultierende Anstieg der Temperatur in den einzelnen Compartments ist in Abb. 6.9 zu sehen. Auf der x-Achse wird die Simulationszeit in Sekunden gezeigt.

Das im Vergleich zur Anfangstemperatur von 20^oC kalte Fenster bewirkt vor dem Hochfahren der Maschine einen Abfall der Temperatur in allen Compartments. Compartment 3 (grüne Linie in Abb. 6.9), in dem sich das Fenster befindet, kühlt hierbei erwartungsgemäß am schnellsten ab. Um 8 Uhr beginnt die Maschine, zu arbeiten und somit Wärme abzugeben, was einen Anstieg der Temperatur in allen Compartments bedeutet. Die schnellste Reaktion auf die Emission findet, wie Abb. 6.9 zeigt, in Compartment 2 (rote Linie), in dem sich die Maschine befindet, statt, gefolgt von Compartment 1 (blaue Linie) und 5 (schwarze Linie), die sich Compartment 2 am nächsten befinden. Compartment 3 bleibt aufgrund des Fensters am kühlsten.

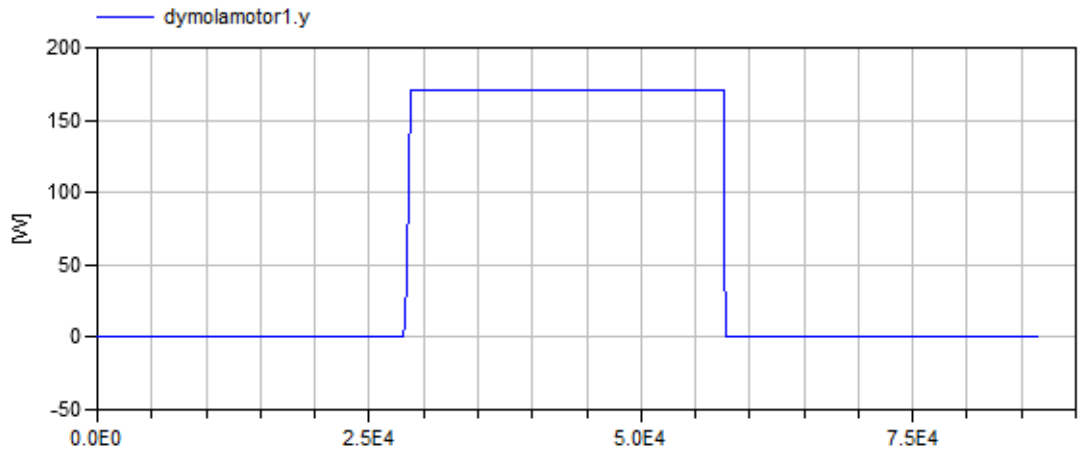

Abbildung 6.8: Vom Gleichstrommotor an die Umgebung abgegebene Wärme für die Simulation über einen Werktag

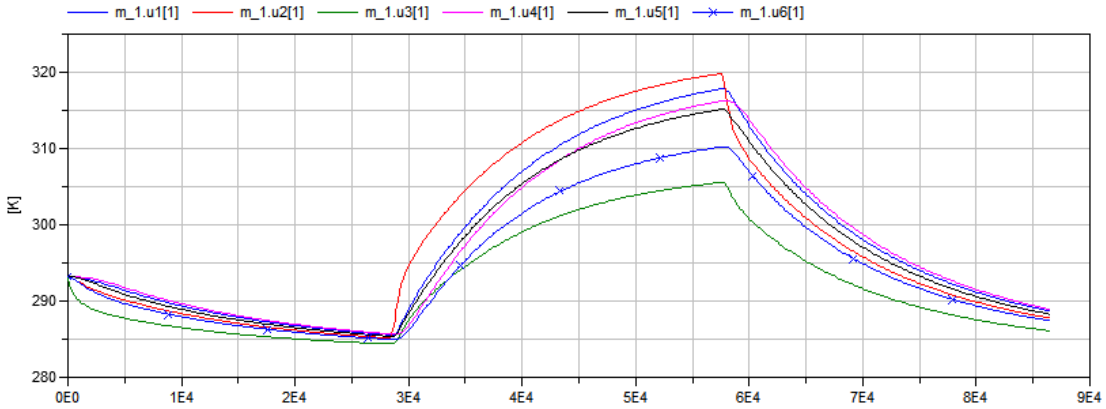

Abbildung 6.9: Reaktion der einzelnen Compartments auf die Wärmeemission des Motors

Kopplung via BCVTB

Das BCVTB-Modell zur Kopplung des Dymola-Raummodells mit dem ebenfalls in Dymola implementierten Maschinenmodell ist in Abb. 6.10 zu sehen.

Abbildung 6.11 zeigt weiters die Hierarchie der Kommunikation zwischen den verwendeten Simulatoren.

Zur Steuerung der Kommunikation wurde ein SDF-Director mit einer fixen Schrittweite von 60 Sekunden verwendet. Zu jedem Synchronisationszeitpunkt erfolgt ein Datenaustausch zwischen den in jeweils eigenen Dymolainstanzen gestarteten Simulationen der beiden Dymolamodelle und der BCVTB-Gesamtsimulation. Die Wärmeemission der Maschine, die bei jedem Schritt als Output des Simulator-Actors namens *motor (dymola)* abgegriffen

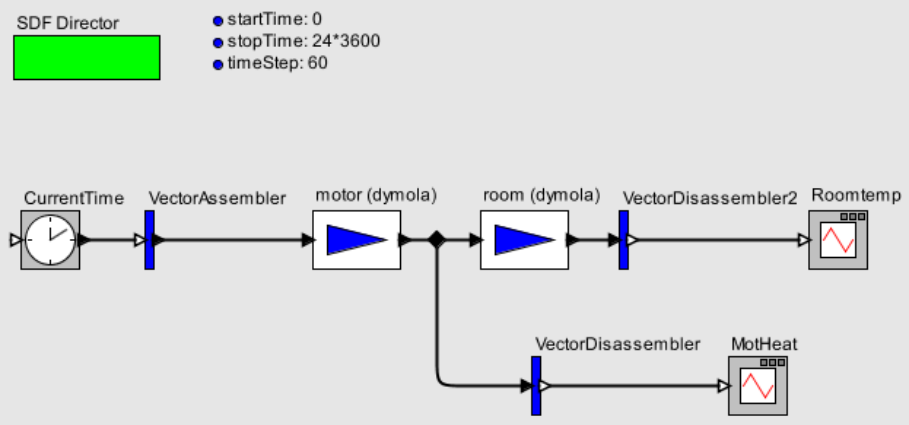

Abbildung 6.10: Modell zur Kopplung der beiden Dymolamodelle via BCVTB

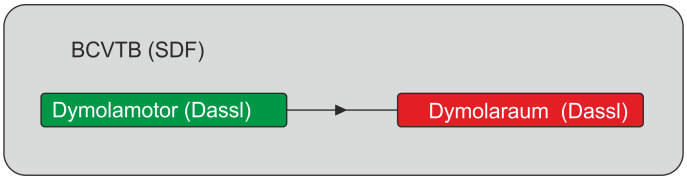

Abbildung 6.11: Veranschaulichung der Co-Simulation zweier Dymolainstanzen über BCVTB

werden kann, wird in BCVTB geplottet sowie dem Simulator Actor für den Raum übergeben. Der Output von Letzterem, welcher der in Compartment 6 gemessenen Temperatur entspricht, wird ebenfalls in einem Plot festgehalten.

Vergleich der beiden Simulationen

Der Temperaturverlauf aus der Kopplung über BCVTB ähnelt jenem der direkten Kopplung in Dymola derart, dass bei der Beobachtung des Verlaufs über den gesamten Tag mit freiem Auge keine Unterschiede erkennbar sind. Zur genaueren Betrachtung ist in Abb. 6.12 die Reaktion auf den Anstieg der Spannung und somit der abgegebenen Wärme des Motors im direkt über Dymola gekoppelten Modell in einem kleineren Intervall um 8 Uhr gezeigt.

Abb. 6.13 zeigt das Äquivalent des über BCVTB gekoppelten Modells.

Der Vergleich dieser beiden Plots zeigt deutlich, dass die Erwärmung in den Compartments bedingt durch das Hochfahren der Maschine in der dymolainternen Simulation schneller erfolgt als in der Co-Simulation über BCVTB. Der Grund für die verzögerte Reaktion in der Co-Simulation ist leicht einzusehen. Da die Synchronisation nur alle 60 Sekunden stattfindet, setzt BCVTB den Wert der letzten Synchronisation innerhalb dieser Intervalle konstant fort, wodurch das Raummodell bis zur folgenden Synchronisation mit einem bis zu 60 Sekunden verspäteten Wert arbeitet. Um diesen Umstand weiter zu verdeutlichen, sind in Abb. 6.14 bzw. 6.15 die Emissionswerte des Motors um 8 Uhr, die im Raummodell

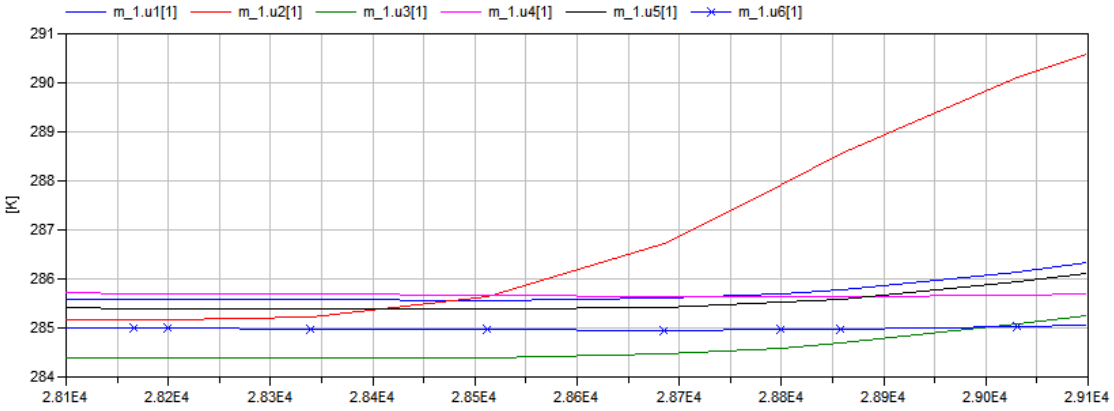

Abbildung 6.12: Temperaturänderung beim Anstieg der an den Motor angelegten Spannung - dymolainterne Kopplung

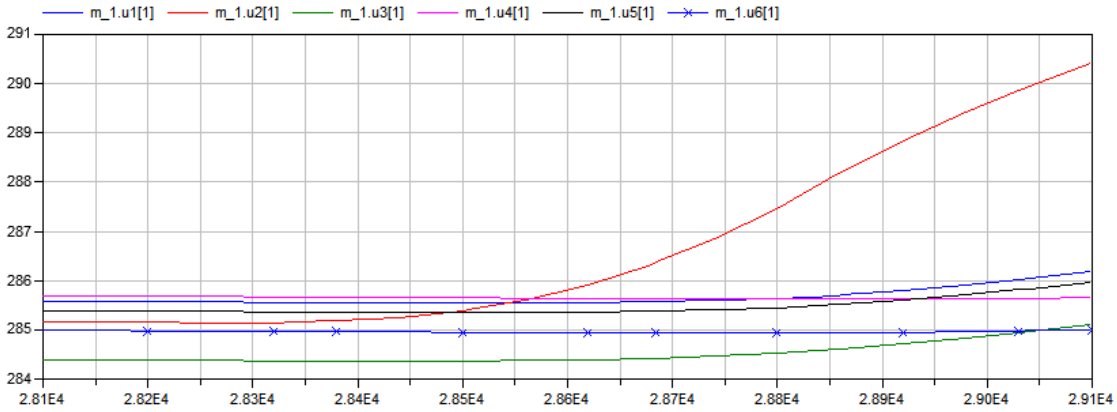

Abbildung 6.13: Temperaturänderung beim Anstieg der an den Motor angelegten Spannung - Kopplung via BCVTB

weiterverwendet werden, gezeigt.

Während im Modell der Gesamtsimulation innerhalb von Dymola die Werte des Wäermeverlusts der Maschine zu jedem Zeitschritt des Solvers, den auch der Raum benötigt, mit berechnet wird, bleibt in der von BCVTB gesteuerten Co-Simulation der Wert während sämtlicher Schritte, die nur vom dymolainternen Solver für das Raummodell benötigt werden, konstant, wodurch die in Abb. 6.15 zu sehende Treppenfunktion entsteht. Da in jeder Zeitspanne, in der die abgegebene Wärme in dieser Simulation konstant gehalten wird, die Wärmeemission in der direkten Kopplung eine Änderung erfahren kann, ist eine verzögerte Reaktion durchaus nachvollziehbar.

Ein weiteres wichtiges Resultat, das es zu vergleichen gilt, sind die von den Solvern getätigten Zeitschritte. Für Dymola wurde für jede Simulation der Solver *dassl* gewählt, da steife Systeme auftreten. Dennoch entstehen aufgrund der Trennung in mehrere Dymolainstanzen

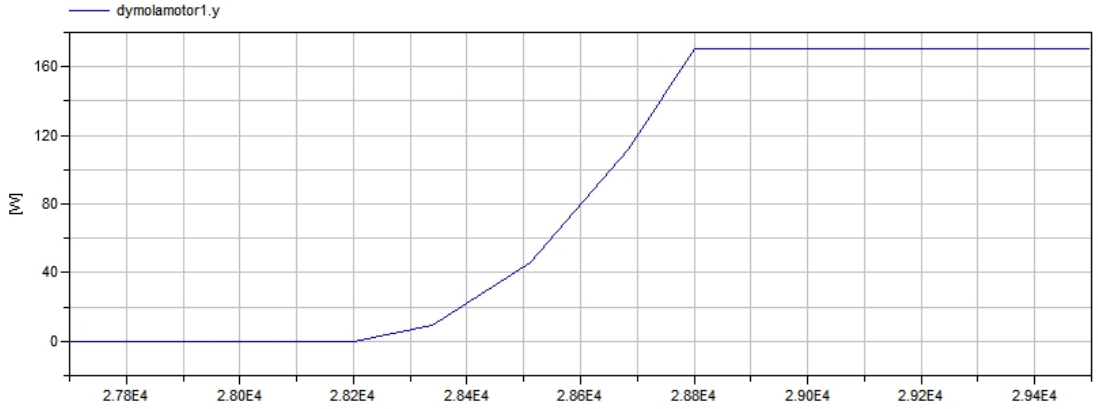

Abbildung 6.14: Temperaturänderung beim Anstieg der an den Motor angelegten Spannung - Kopplung via Dymola

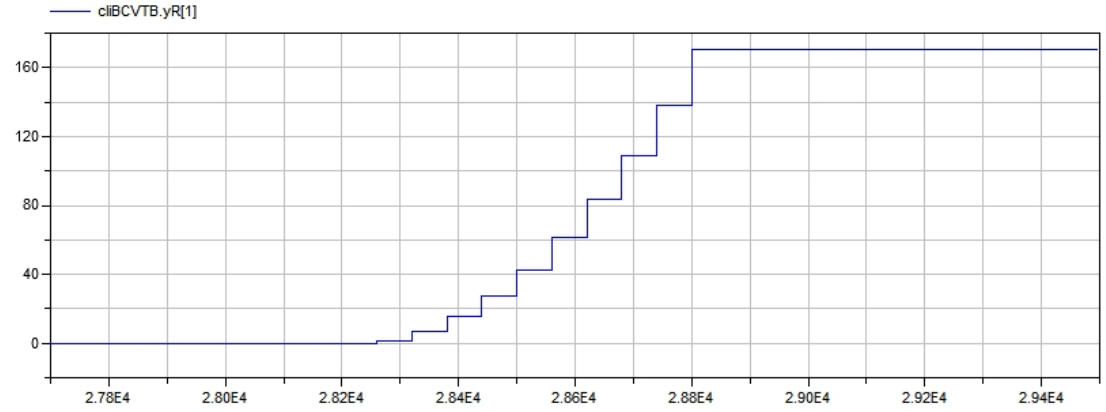

Abbildung 6.15: Temperaturänderung beim Anstieg der an den Motor angelegten Spannung - Kopplung via BCVTB

bei der Kopplung über BCVTB Unterschiede zur Simulation in einer einzigen Dymolainstanz, wie die Plots der getätigten Solver-Zeitschritte in weiterer Folge zeigen werden. In Abb. 6.16 werden die Schritte, die Dassl in der gemeinsamen dymolainternen Simulation verwendet, in einem Intervall, das kurz vor Arbeitsbeginn des Motors beginnt, gezeigt. Zur besseren Veranschaulichung der Schrittweiten sind die Übergänge von einem Zeitschritt zum nächsten als ellipsenförmige Sprünge dargestellt.

Da zehn Minuten vor acht Uhr, was der Simulationszeit von 28200 Sekunden entspricht, die Maschine hochzufahren beginnt, was einen plötzlichen Anstieg der Spannung und somit der Wärmeemission der Maschine zur Folge hat, muss der Solver auf dieses Event mit vorübergehend sehr kleinen Zeitschritten reagieren.
Abb. 6.17 zeigt die Zeitschritte der beiden verwendeten Dassl-Solver sowie den Zeitschritt für die Gesamtsimulation vom SDF-Director im selben Simulationszeitintervall.

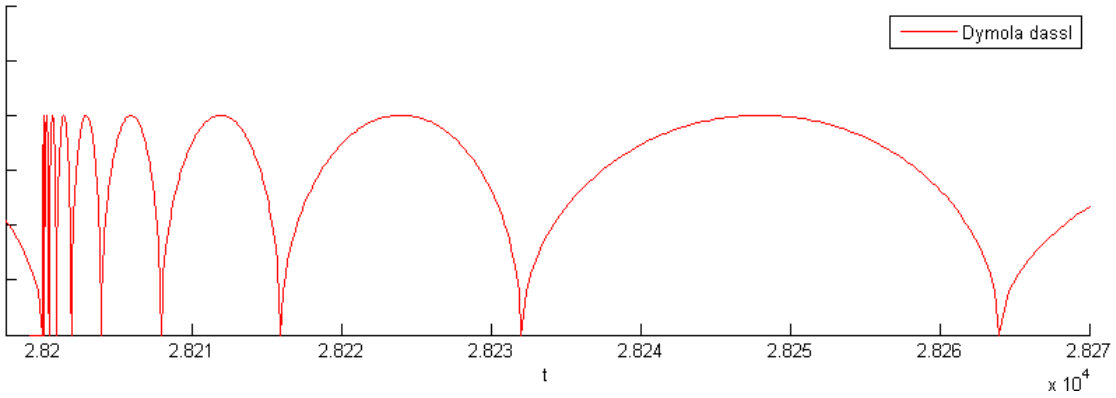

Abbildung 6.16: Solverzeitschritte von Dassl bei der dymolainternen Simulation

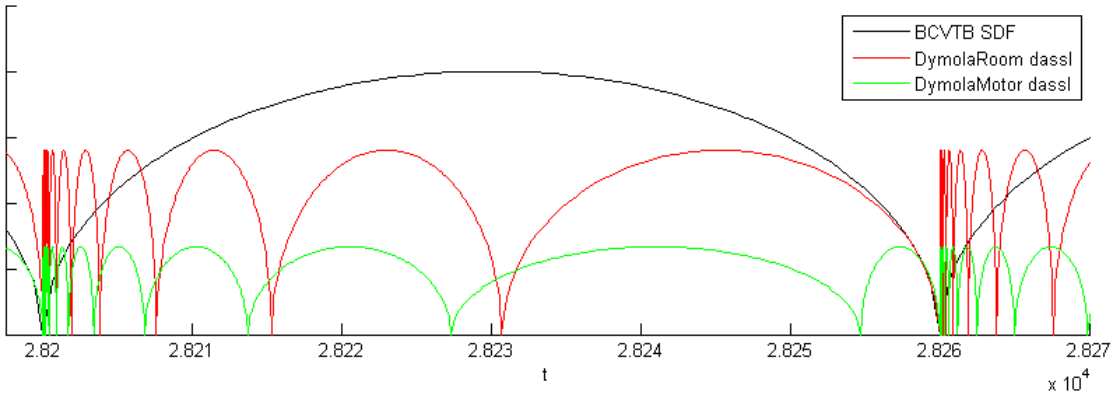

Abbildung 6.17: Solverzeitschritte bei der Co-Simulation

Eine Tatsache, die sofort ins Auge springt, sind die - trotz Verwendung des gleichen Solvers - unterschiedlich großen Zeitschritte für die Simulation des Motormodells im Vergleich zu dem Raummodell. Da im Motormodell neben thermischen Vorgängen auch elektrische sowie mechanische Zustandsänderungen mitsimuliert werden, benötigt der Solver zur Einhaltung eines maximalen Fehlers kleinere Zeitschritte, als in der Simulation des Raummodells notwendig sind, in dem lediglich thermische Vorgänge, die per se in größeren Zeitkonstanten agieren, auftreten.
Dies würde zunächst die Vermutung unterstützen, dass die Co-Simulation durch die Trennung der Einzelmodelle und der daraus resultierenden Vermeidung der oftmals redundanten Berechnung der Zustände in der Simulation des Raummodells eine schnellere Simulation mit einer wesentlich geringeren Anzahl an getätigten Zeitschritten zur Folge hat.
Der dokumentierte CPU-Zeitverbrauch für die Simulationszeit von einem Tag ergibt jedoch 16.9 Sekunden bei der Simulation durch eine Dymolainstanz gegenüber 18.6 Sekunden für die Co-Simulation via BCVTB, was diese Vermutung sofort widerlegt. Der Grund dafür ist bei der Beobachtung der Solver-Zeitschritte zum nächsten Synchronisationszeitpunkt bei 28260

Sekunden deutlich erkennbar. Da nach einer Synchronisation mit BCVTB alle Zustände, die seit der letzten Synchronisation nicht konstant geblieben sind, einen Sprung erfahren (siehe etwa Abb.6.15), reagieren die Solver der gekoppelten Simulatoren mit neuerlicher Schrittweitenadaptierung und somit einer Unzahl an kleinen Zeitschritten nach jeder Synchronisation. Dies resultiert in einer insgesamt höheren CPU-Zeit für die Co-Simulation. Als Fazit aus den Beobachtungen dieses Kapitels lässt sich schließen, dass eine Co-Simulation von Modellen, die in derselben Software implementiert sind, erst für die Kopplung von sehr komplexen Einzelmodellen interessant wird.

6.1.1.3 Kopplung des Raummodells mit einer in Simscape implementierten Maschine

Dieser Abschnitt dokumentiert die Simulation des in bereits in Kapitel 6.1.1.2 beschriebenen Raumes mit einem in Simscape implementierten Gleichstrommotor (siehe Kapitel 5.4). Ein Überblick über die beabsichtigte Kommunikation dieser Simulatoren ist in Abb. 6.18 zu sehen.

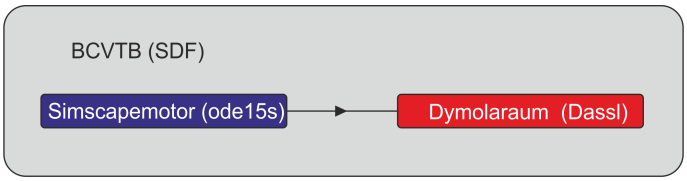

Abbildung 6.18: Übersicht der Kommunikation zwischen den Simulatoren BCVTB, Simscape und Dymola

Die Werte für die einzelnen Komponenten des Motors wurden zu Vergleichszwecken gleich gewählt, wie sie im in Dymola verwirklichten Motormodell verwendet wurden. Neben dem Vergleich der beiden Implementierungs- sowie Simulationsmöglichkeiten mit Dymola/Modelica und Simulink wird sich der Beginn dieses Abschnitts damit beschäftigen, die Unterschiede der Solverzeitschritte und somit des CPU-Zeitverbrauchs bei iterierter Synchronisationszeit im Gegensatz zur Verwendung einer im Vorhinein gesetzten Sample Time zu erörtern.

Vergleich der Simulationen des Simscape-Maschinenmodells mit jener des Dymola-Maschinenmodells

Bei der Wahl derselben Parameter für die einzelnen Simulationen kann bezüglich des Verlaufs der Outputs kein Unterschied festgestellt werden, wie Abb. 6.19 sowie 6.20 veranschaulichen.

Sowohl die emittierte Wärme der Maschinen und in deren Folge auch der Temperaturverlauf in den Compartments stimmen überein, was dem Anspruch nach Konsistenz bei der Verwendung von unterschiedlichen Modellierungsumgebungen und Simulatoren bei der gleichen Wahl der Parameter genügt. In Abb. 6.20 wurde Compartment 4, dessen Temperaturoutput in jedem Zeitschritt zu BCVTB transferiert wird, als Repräsentant gewählt.

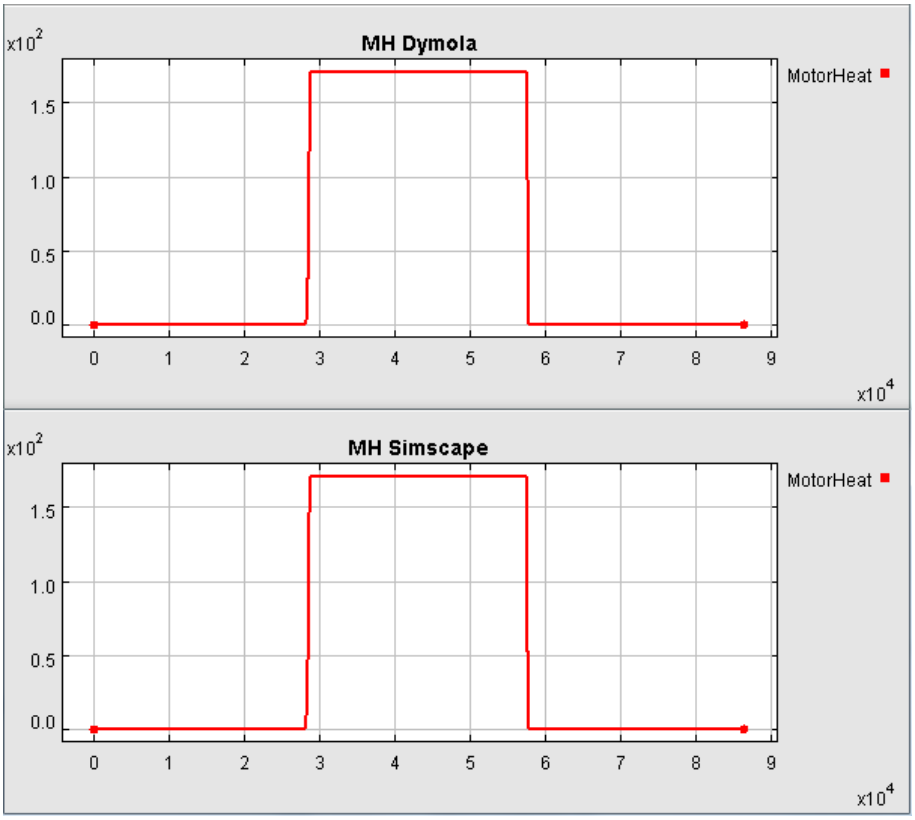

Abbildung 6.19: Vergleich der resultierenden Wärmeemission des Motormodells, das zum einen in Dymola, zum anderen in Simscape implementiert und simuliert wurde

Dennoch kann im Verlauf der Simulation beobachtet werden, dass die Solver *Dassl* von Dymola und *ode15s* aufgrund ihrer individuellen Algorithmen ein völlig anderes Verhalten im Setzen von Zeitschritten vorweisen. Für die Gesamtsimulation über BCVTB wurden in beiden Fällen 60 Sekunden als fixer Zeitschritt des SDF-Directors und somit Synchronisationszeitpunkt gewählt.
Abb. 6.21 zeigt die unterschiedlich gesetzten Zeitschritte der Solver für die Simulation der Motoren. Zur Orientierung wurde zusätzlich die Synchronisationsschrittweite von 60 Sekunden eingezeichnet. Da sich der Dassl-Solver für die Simulation des Raummodells von der Wahl des Simulators für den Motor unbeeindruckt zeigt, sind dessen Zeitschritte in diese Graphik nicht mit einbezogen.

Abb. 6.22 kann entnommen werden, dass Dassl nach der Synchronisation auch zurückspringen muss, um mit einer adäquaten Schrittweite weiterzurechnen, allerdings benötigt ode15s von Simscape deutlich mehr redundante Schritte für die neuerliche Steuerung der Schrittweite nach der Synchronisation.

Neben der eben gezeigten Gegenüberstellung der Schrittweiten der beiden Solver für die Motormodelle kann die verbrauchte CPU-Zeit der beiden Simulationen verglichen wer-

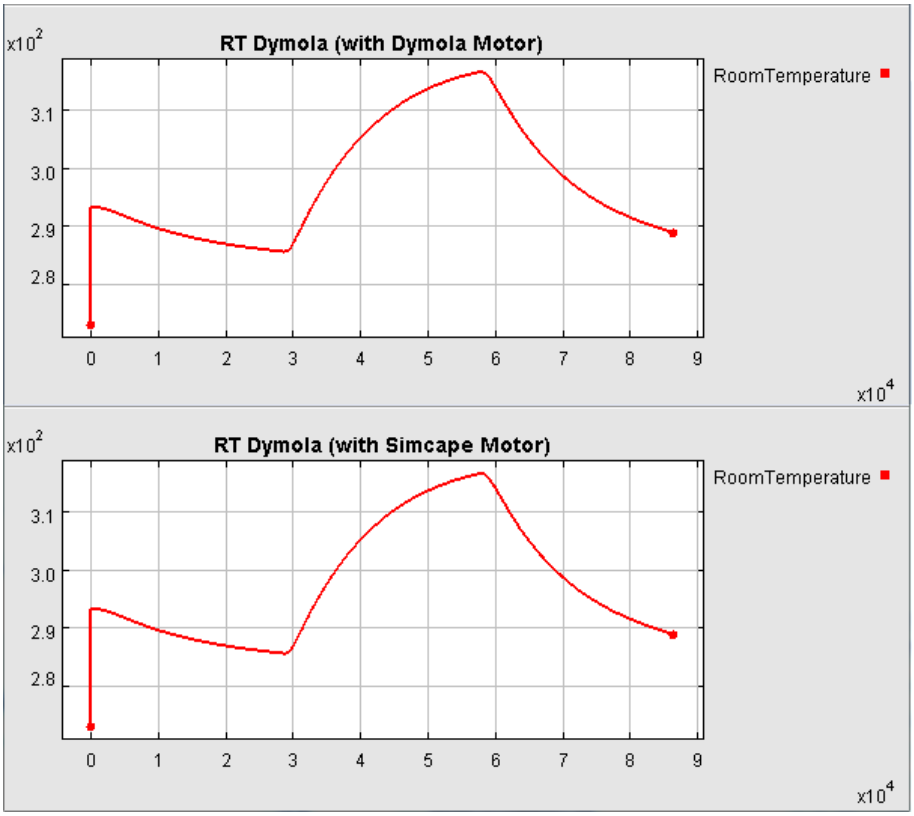

Abbildung 6.20: Vergleich der resultierenden Raumtemperatur bei Verwendung des einerseits in Dymola, andererseits in Simscape implementierten und simulierten Motormodells

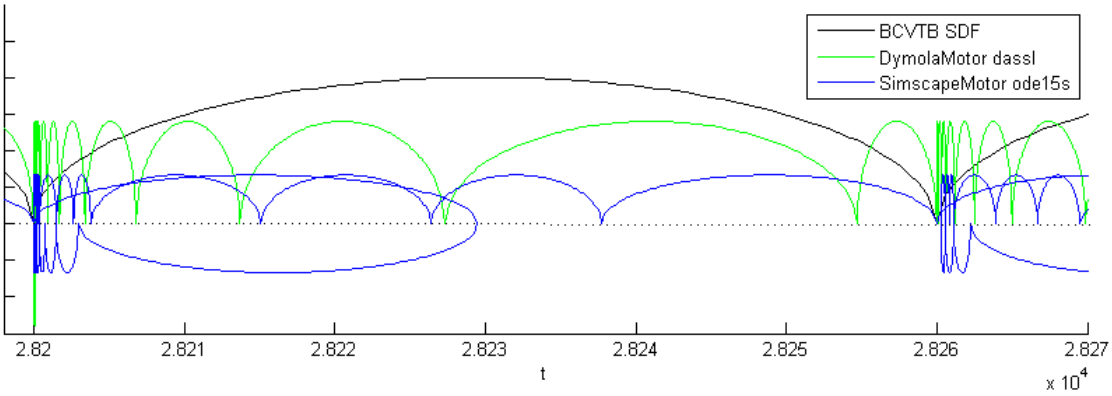

Abbildung 6.21: Vergleich der Zeitschritte bei der Co-Simulation der Solver *Dassl* für Dymola bzw. *ode15s* für Simscape

den. Um stichhaltige Aussagen über die Effizienz zu erhalten, wurden beide Modelle mehr-

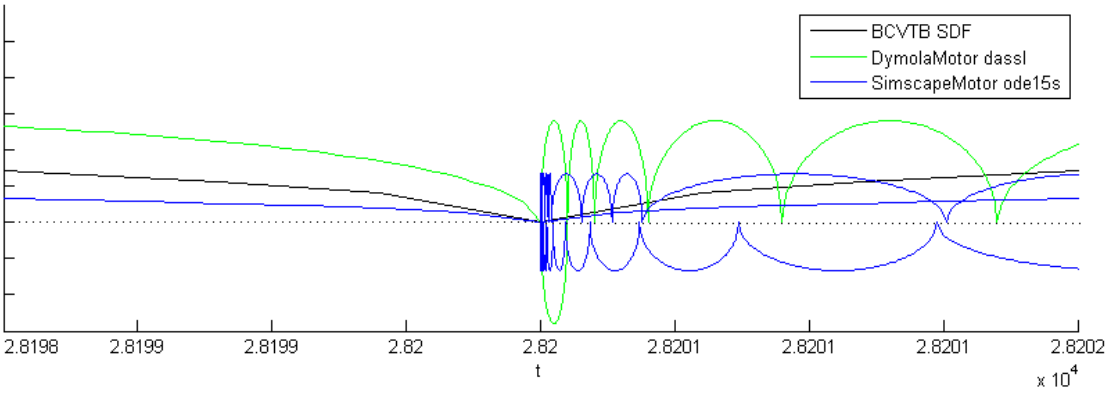

Abbildung 6.22: Vergleich der Zeitschritte der Solver *Dassl* für Dymola bzw. *ode15s* für Simscape zum Synchronisationszeitpunkt

mals simuliert und die Durchschnittswerte ermittelt. Dies ließ die Tendenz erkennen, dass die Co-Simulation des Dymola-Motormodells mit dem Raum über einen Tag mit 18.6 Sekunden im Mittel lediglich um 0.4 Sekunden schneller ist als das entsprechende Experiment mit dem Simscape-Motormodell. Dieser beinahe vernachlässigbare Unterschied kann dadurch erklärt werden, dass die größere Anzahl an Solverzeitschritten für die Schrittweitensteuerung von ode15s nach einer Synchronisation durch die größeren und somit wenigeren Schritte im Verlauf der Simulation außerhalb der unmittelbaren Nähe von Synchronisationszeitpunkten kompensiert wird.

Vergleich der Synchronisation mit bzw. ohne Iteration des Simscape-Solvers

In Kapitel 5.1.3 wurde bereits erwähnt, dass zur Synchronisation mit BCVTB zu einem gewünschten fixen Zeitschritt überprüft wird, wann die Simulationszeit in Simscape ein ganzzahliges Vielfaches dieses Wertes überschreitet. Da der Solver ohne weitere Einstellungen im Allgemeinen keine Zeitschritte im gewünschten Toleranzintervall um diese Zeitpunkte setzt, muss die Überschreitung in diesem Fall iteriert werden. Durch das Setzen einer *Sample Time* in einem beliebigen verwendeten Block des Modells und die Deaktivierung der Iterationsmöglichkeit in dem Block zum Vergleich der Simulationszeiten können diese Iteration und somit eine Vielzahl an redundanten Auswertungen verhindert werden.
Zur Verdeutlichung dieses Umstands wurde das oben beschriebene Modell einmal ohne Definition einer Sample Time im Simscape-Motormodell und einmal mit dem Setzen einer Sample Time auf den im SDF-Director von BCVTB verwendeten Zeitschritt von 60 Sekunden simuliert. Die Zeitschritte, die der Solver, jeweils ode15s, in beiden Simulationen zur selben Simulationszeit setzt, sind in Abb. 6.23 zwischen zwei Synchronisationszeitpunkten gezeigt.
Die hellblaue Linie stellt, wie unschwer zu erkennen ist, die Zeitschritte bei iteriertem Synchronisationszeitpunkt dar. Die Schritte des Solvers bei Verwendung einer Sample Time entsprechen der dunkelblauen Linie. Zusätzlich sind wiederum die fixen Zeitschritte des SDF-Directors von BCVTB in schwarzer Farbe eingezeichnet, um die Synchronisationszeit-

punkte hervorzuheben.

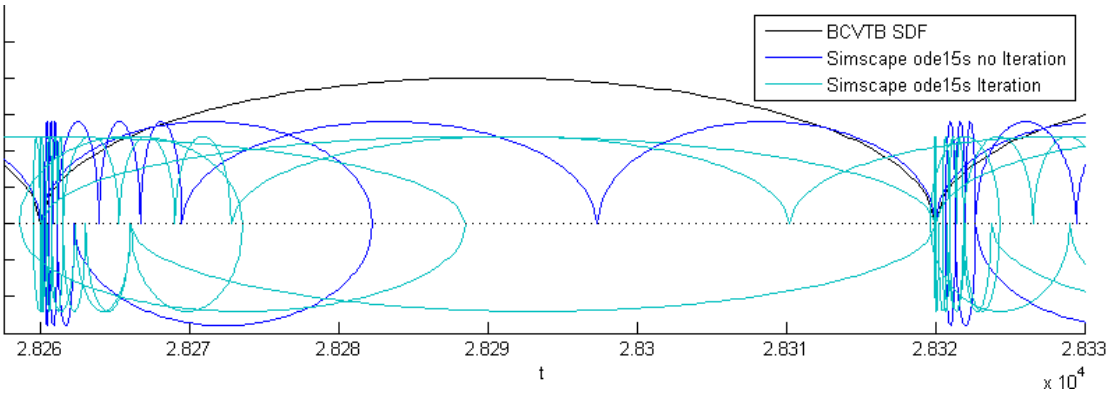

Abbildung 6.23: Vom Simscape-Solver ode15s getätigte Zeitschritte mit und ohne Verwendung einer Sample Time

Bei der Betrachtung von Abb. 6.23 kann sofort festgestellt werden, dass die Vermeidung des auf Anhieb exakten Erreichens der Synchronisationszeit auch Einfluss auf das Verhalten des Solvers in der weiteren Simulation hat. So folgen beispielsweise dem Synchronisationszeitpunkt bei 28260 Sekunden in der Simulation mit Synchronisationszeititeration sowohl mehr im Nachhinein verworfene Zeitschritte zur erneuten Ermittlung der optimalen Schrittweite für den hinter dem Solver stehenden Algorithmus (zu erkennen an den Bögen, die in der unteren Halbebene zurückspringen) als auch kleinere Schritte im weiteren Verlauf.
Abb. 6.24 zeigt den ursprünglich zu erwartenden und zugleich markantesten Unterschied im Verhalten des Solvers in den beiden Simulationen in einem Intervall von insgesamt 4 Sekunden um einen Synchronisationszeitpunkt. Die hellblauen Bögen hüpfen zunächst weit über den Synchronisationszeitpunkt bei 28260 Sekunden. Dem Erkennen dieser Überschreitung folgt ein Zurückspringen des Solvers bis zur neuerlichen Kreuzung des zu iterierenden Zeitpunkts. Diese Vorgehensweise wird bei fortlaufender Verkleinerung der Schritte wiederholt, bis ein Zeitpunkt innerhalb des vorgegebenen Toleranzintervalls um die gewünschte Synchronisationszeit erreicht wird. Im Gegensatz dazu findet sich kein dunkelblauer Bogen, der von jedweder Richtung den Synchronisationszeitpunkt überschreitet. Der letzte Schritt vor der Synchronisation wird vom Solver exakt so gewählt, dass zum gewünschten Zeitpunkt eine Berechnung erfolgt, was der einzige, genau bei 28260 Sekunden von links eintreffende dunkelblaue Bogen deutlich zeigt.

Beim Vergleich der verbrauchten CPU-Zeit kann trotz der zahlreichen überflüssigen Zeitschritte des Solvers bei der Iteration lediglich ein um 0.9 Sekunden höherer Zeitverbrauch pro Simulationstag im Gegensatz zur Simulation bei Aktivierung einer Sample Time festgestellt werden. Dieser Aspekt spricht dafür, auch die Möglichkeiten bei Verwendung eines Continuous Directors zur Gesamtsimulation weiter zu verfolgen. Bei einer solchen Simulation kann die Synchronisationszeit selbstverständlich nicht im Vorhinein gesetzt werden, da sie variabel und auch schlichtweg nicht bekannt ist. Somit wäre eine Iteration zur Synchro-

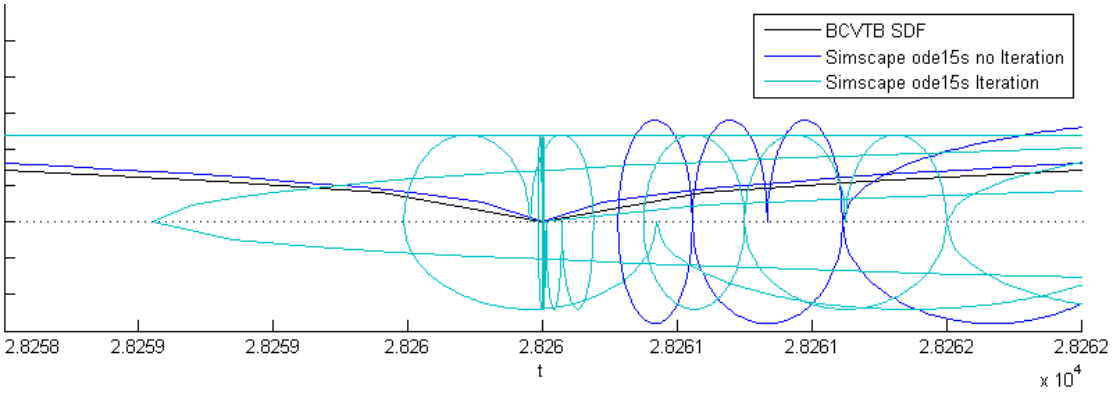

Abbildung 6.24: Genaues Erreichen vs. Iteration des Synchronisationszeitpunkts des Simscape-Solvers ode15s

nisation unumgänglich.

6.1.1.4 Versuch der Simulation des Simscape-Motormodells über BCVTB mit einem Continuous Director

In Kapitel 5.1.3 wurde erwähnt, dass es möglich ist, Simscape zu variablen Zeitpunkten mit BCVTB kommunizieren zu lassen. Wie die folgenden beiden Experimente zeigen, ist dies in der Praxis nur unter drastischen Einschränkungen realisierbar. In Abbildung 6.25 ist die Schachtelung der Simulatoren mit den verwendeten Solvern dargestellt.

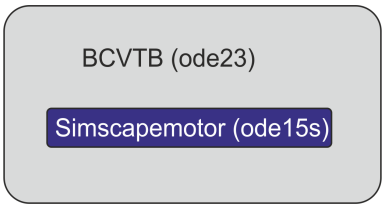

Abbildung 6.25: Veranschaulichung der Kommunikation zwischen den Simulatoren Simscape und Dymola über BCVTB

Abbildung 6.26 zeigt ein BCVTB-Modell mit einem Continuous Director für die Gesamtsimulation, das lediglich die Kommunikation mit Simscape ermöglicht und den Output der Simulation des Motormodells, welcher der Wärmeleistung entspricht, in einem Timed Plotter von BCVTB darstellt.

Für den Continuous Director wurde ode23 als Solver gewählt, dessen maximale Schrittweite auf 10 Sekunden gesetzt wurde. Für ode15s von Simscape wurden ebenfalls 10 Sekunden als maximale Schrittweite gewählt. Die Simulation erfolgte über 6 Minuten, wobei nach 3 Minuten und 20 Sekunden das Hochfahren des Motors durch die Erhöhung der an-

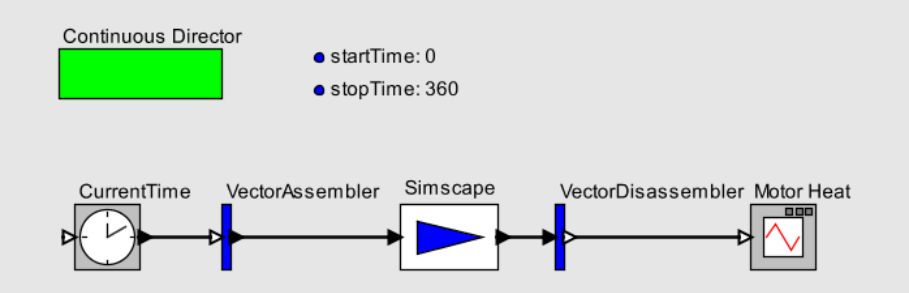

Abbildung 6.26: BCVTB-Modell für die Simulation des Simscape-Motormodells über einen Continuous Director mit ode23 als darüberstehendem Solver

gelegten Spannung begann.
Die Schritte, die von beiden Solvern in den ersten 20 Sekunden getätigt wurden, zeigt Abb. 6.27. Die schwarzen Bögen stellen die Zeitschritte des Continuous Directors dar, die blaue Linie zeigt die von ode15s getätigten Zeitschritte. Da sich der Motor zu Beginn im Ruhezustand befindet, würde Simscape mit einer hohen Schrittweite beginnen. Die anfangs kleinen Schritte des Continuous Directors verlangen jedoch nach häufiger Synchronisation, was viele redundante Schritte zur Iteration in Simscape bewirkt.

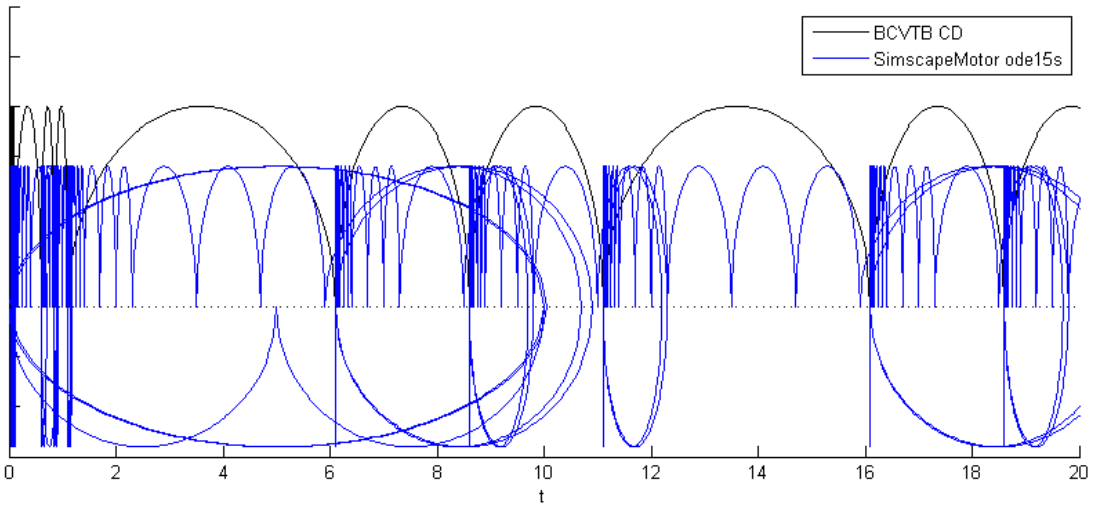

Abbildung 6.27: Zeitschritte des Continuous Directors und des Simscape-Solvers ode15s zu Beginn der Simulation

Nach der anfänglichen Verwendung relativ kleiner Schritte beider Solver kann für den Continuous Director schon nach kurzer Zeit eine Regelmäßigkeit festgestellt werden, die ob der ausbleibenden Veränderung im Gesamtmodell durchaus nachvollziehbar ist. Auch die Erhöhung der Spannung bewirkt lediglich eine Verkleinerung der Schritte des Simscape-Solvers selbst (siehe Abb.6.28), die für die Häufigkeit des Datenaustauschs jedoch irrelevant

sind und demnach keine Auswirkungen auf die Schrittweite in der Gesamtsimulation haben.

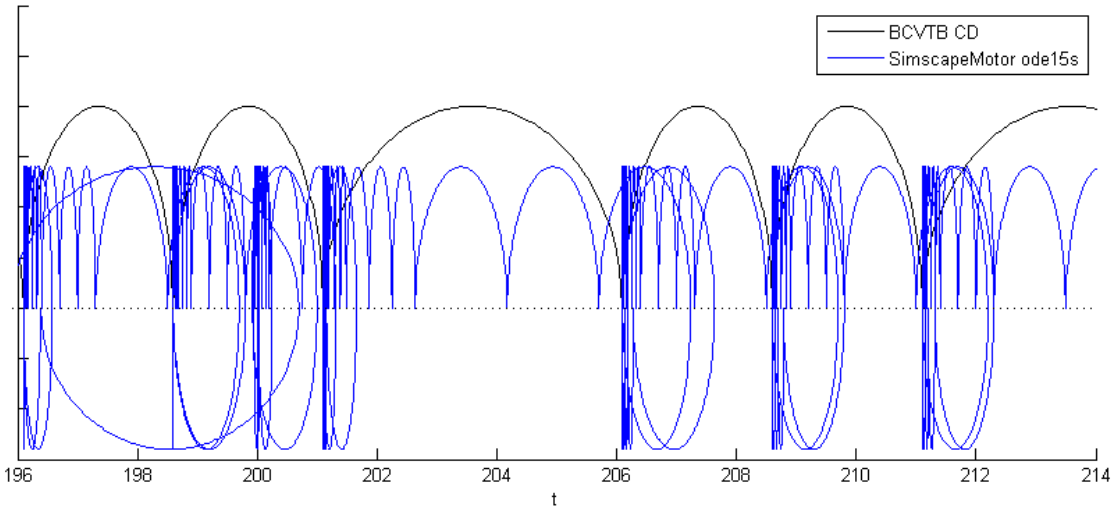

Abbildung 6.28: Zeitschritte des Continuous Directors und des Simscape-Solvers ode15s beim Hochfahren des Motors

Um die Zeitschritte des Continuous Directors stärker zu variieren, wurde dem BCVTB-Gesamtmodell ein Integratorblock hinzugefügt, wodurch aus der abgegebenen Wärmeleistung die frei gewordene Energie erhalten werden kann. Bis zu dem Zeitpunkt, an dem die Spannung am Motor erhöht wird und somit der Output sowie dessen Integral konstant Null beträgt, treten keine Probleme und ein ähnliches Verhalten beider Solver zur Simulation des vorhergehenden Modells auf. Sobald jedoch die Wärmeleistung zu steigen beginnt, was der Continuous Director beim ersten Zeitschritt nach 200 Sekunden erkennt, versucht der Solver ode23, zu 193 Sekunden zurückzuspringen, um die Schrittweite zur adäquaten Miteinberechnung dieser Veränderung doch kleiner zu wählen. Da der Solver in Simscape nicht mehr zurückspringt, läuft dort die Simulationszeit immer weiter. Die Differenz zwischen dieser Zeit und dem kleineren Wert von BCVTB wächst somit auch immer weiter und hat nie wieder einen Nulldurchgang; somit kann dieser auch nicht iteriert werden, geschweige denn eine Aktivierung der Kommunikation mit BCVTB bewirken. Als Resultat daraus ergibt sich ein von BCVTB unbeeindrucktes Weiterlaufen der Simulation in Simscape und ein nicht enden wollendes Warten von BCVTB auf die Antwort von Simscape, also ein Deadlock aus Sicht von BCVTB.
Da somit jeder Schritt des Solvers für die Gesamtsimulation in negative Zeitrichtung für einen erfolgreichen Ablauf der Simulation unzulässig ist, dürften nur derart einfache Modelle die Schrittweite beeinflussen, durch welche diese so regelmäßig bleibt, dass ohnehin ein Solver mit fixer Schrittweite vorzuziehen wäre, der zudem keine Iteration des Synchronisationszeitpunktes erfordert.

6.1.1.5 Kopplung des Raummodells mit dem Modell einer in Simscape implementierten, komplizierten Werkzeugmaschine

Um festzustellen, wie die von BCVTB gesteuerte Co-Simulation mit Systemen von hoher Komplexität und Steifigkeit umgeht, wurde das Dymola-Raummodell mit einer Zwischenstufe des Schneidemaschinenmodells, das im Zuge von [Hei12] entwickelt wurde und in Kapitel 5.5 beschrieben ist, über BCVTB gekoppelt. Aufgrund des signifikant kleineren Zeitschrittes, der zur Simulation dieses Maschinenmodells im Gegensatz zu den vorhergehenden Beispielen notwendig ist, sowie der geringeren Gesamtsimulationszeit von einer Minute, die einem Schneidevorgang entspricht, wurde auch das Synchronisationsintervall mit 0.5 Sekunden demgemäß angepasst. Das Zusammenspiel der Solver entspricht erneut der Übersicht aus Abb.6.25.
Abbildung 6.29 zeigt die Wärmeabgabe der Maschine sowie die Reaktion der Raumtemperatur auf diese.

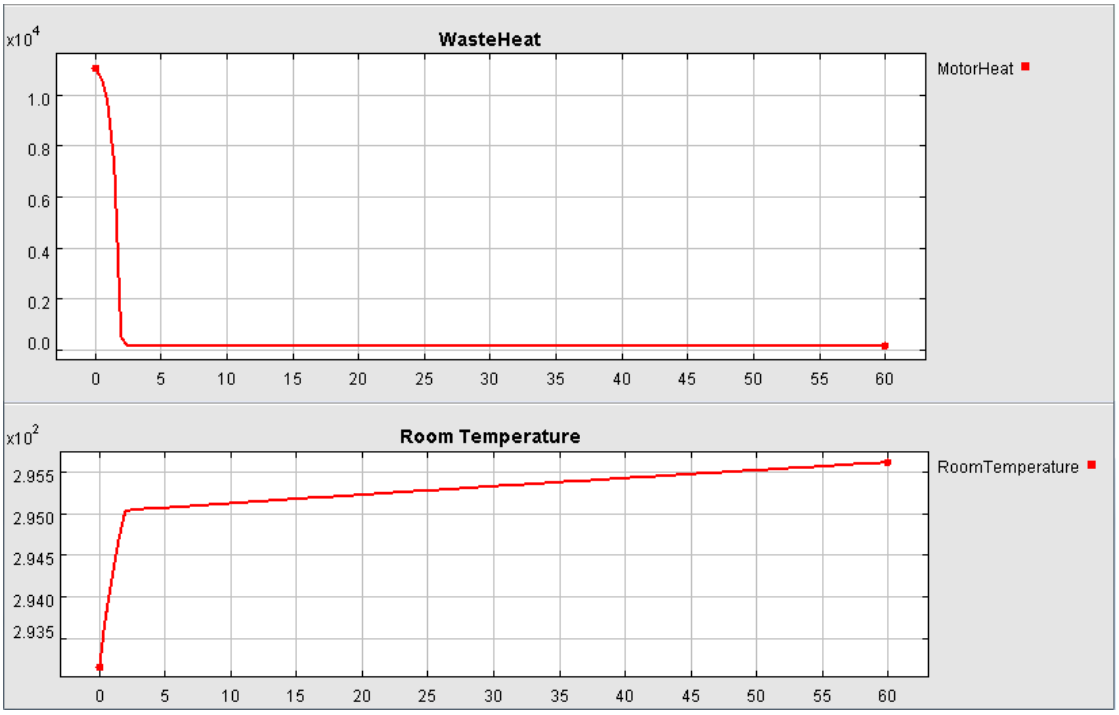

Abbildung 6.29: Wärmeabgabe der Werkzeugmaschine und Reaktion der Raumtemperatur

Nach dem Hochfahren der Maschine wird zunächst sehr viel Wärme abgeben, was durch den Aufwand zur Überwindung des Trägheitsmoments zu erklären ist, da damit ein hoher Drehmoment- und somit Strombedarf des Motors verbunden ist, der große ohmsche Verluste bewirkt. Danach bleibt die Wärmeemission in etwa konstant bei 140 Watt, wodurch nach der schnellen Erwärmung zu Beginn ein weitaus langsamer Anstieg der Temperatur erfolgt.
Die Zeitschritte beider Simulatoren sind in Abb. 6.30 dargestellt. Da die Zeitschritte vom Sol-

ver ode15s so klein gewählt werden, dass in einem ganzen Synchronisationsintervall kaum etwas zu erkennen wäre, wurde nur ein Intervall von 0.01 Sekunden nach einem Synchronisationszeitpunkt dargestellt.

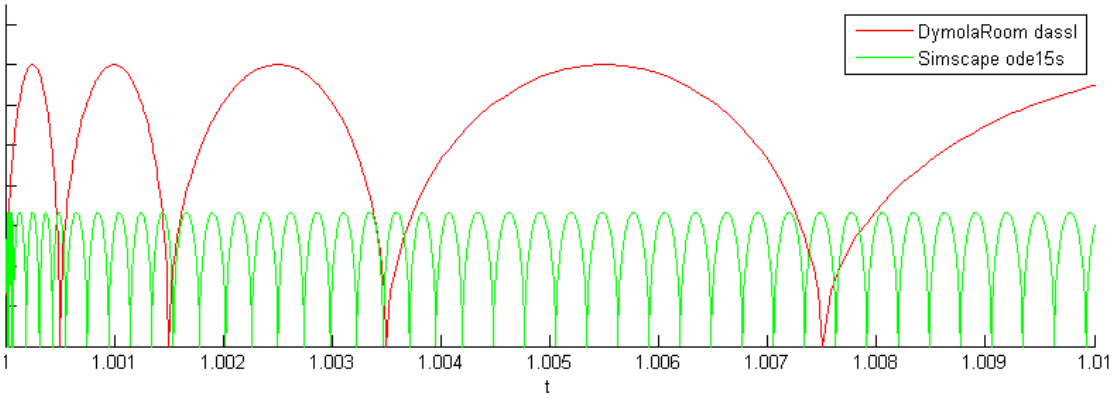

Abbildung 6.30: Zeitschritte des Dymola-Solvers Dassl und des Simscape-Solvers ode15s

Durch die kleinen Zeitschritte des Simscape-Solvers ode15s, die für die Simulation dieser komplexen Maschine notwendig sind, beläuft sich die CPU-Zeit für diese 60 Sekunden Simulationszeit auf 83.4 Sekunden. Dies zeigt, dass diese Simulation sogar langsamer als in Echtzeit läuft, was für die Simulation eines Gesamtmodells über einen Zeitraum von Monaten oder Jahren natürlich jeder Rechtfertigung entsagt. Im Fall einer solchen Maschine, deren Verhalten durch die Außenbedingungen nicht beeinflusst wird, ist es demnach jedenfalls zielgerichteter, das Maschinenmodell zunächst über einen kurzen Zeitraum separat zu simulieren, die Daten für die Wärmeabgabe aufzuzeichnen und für jeden Arbeitszeitraum der Maschine periodisch fortzusetzen, um sie so dem Raummodell in einer Simulation über einen langen Zeitraum als schlichtes Datenmodell zuführen zu können.

6.1.1.6 Kopplung des Raummodells mit mehreren Maschinenmodellen sowie einer Temperaturregelung

Das folgende Experiment wurde zur Demonstration der Co-Simulation mehrerer verschiedener Simulatoren gewählt. Zunächst wurde wieder ein Raummodell in Dymola verwirklicht, das aus 12 Luftcompartments mit den Abmessungen $2m \times 2m \times 3m$ besteht und somit eine Maschinenhalle mit einer Gesamtgröße von $4m \times 6m \times 6m$ repräsentiert. Eine Skizze dieses Raumes kann in Abb. 6.31 betrachtet werden.

In der Maschinenhalle sollen sich drei Maschinen befinden, die mithilfe von Simscape, Dymola und MATLAB modelliert wurden. Die von diesen eingehenden Wärmeströme sind in Abb. 6.31 mit *Q-sim*, *Q-dym* bzw. *Q-mat* bezeichnet. Die Modelle der einzelnen Maschinen sind in den Kapiteln 5.4, 5.6 und 5.7 beschrieben. *Q-ctrl* steht in der Skizze für die Zu- oder Abfuhr von Wärme durch das Temperaturregelungssystem. Das beabsichtigte Zusammenspiel aller verwendeten Simulatoren für die Gesamtsimulation dieses Maschinenraumes über BCVTB ist in Abb. 6.32 veranschaulicht.

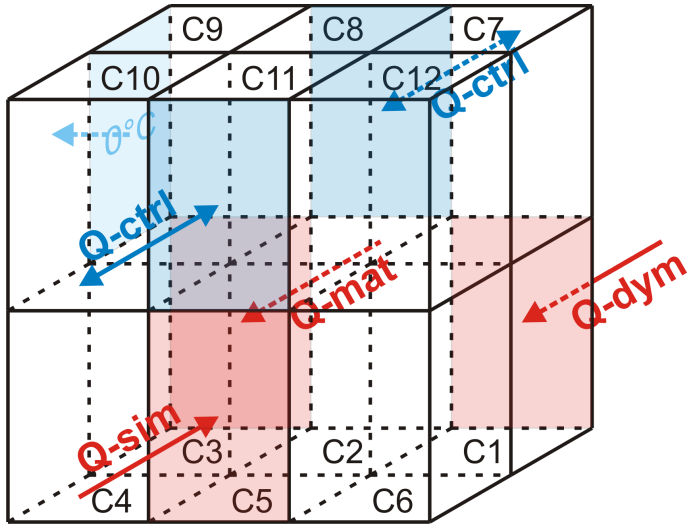

Abbildung 6.31: Veranschaulichung des verwendeten Raummodells, dessen Unterteilung sowie der ein- und ausgehenden Wärmeströme

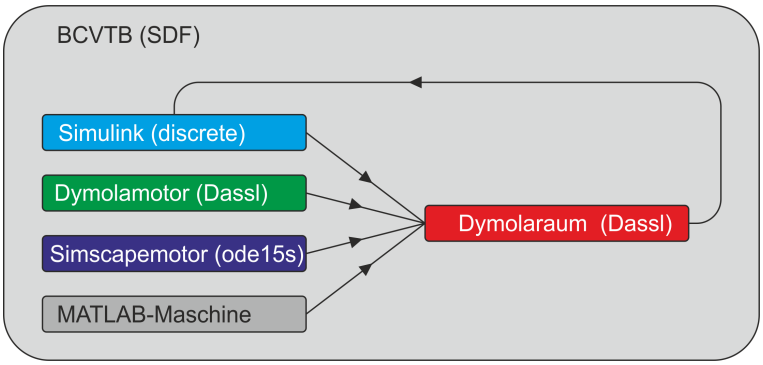

Abbildung 6.32: Beabsichtigte Kommunikation von insgesamt 3 MATLAB-Instanzen und 2 Dymola-Instanzen über BCVTB

Als Parameter für das in Simscape implementierte Modell des Gleichstrommotors wurden $R = 250\Omega$, $L = 16 \cdot 10^{-3}H$ und $k = 0.9Nm/A$ gewählt. Der Motor beginnt um 07:45 Uhr auf den Arbeitsbetrieb mit $230V$ hochzufahren und erreicht diesen nach 15 Minuten. Bis 16:00 Uhr bleibt die Spannung konstant, wonach sie innerhalb von 7.5 Minuten wieder in den Ruhezustand von $0V$ gelangt. Die durch dieses Modell repräsentierte Maschine soll sich im Gesamtmodell in Compartment 5 befinden, was im Modell realisiert wird, indem die Wärmeabgabe dieses Motors an BCVTB, von BCVTB an Dymola und dort in weiterer Folge als vorgeschriebener Wärmefluss einem *HeatPort* des Compartments 5 übergeben wird (siehe Abb. 6.33).

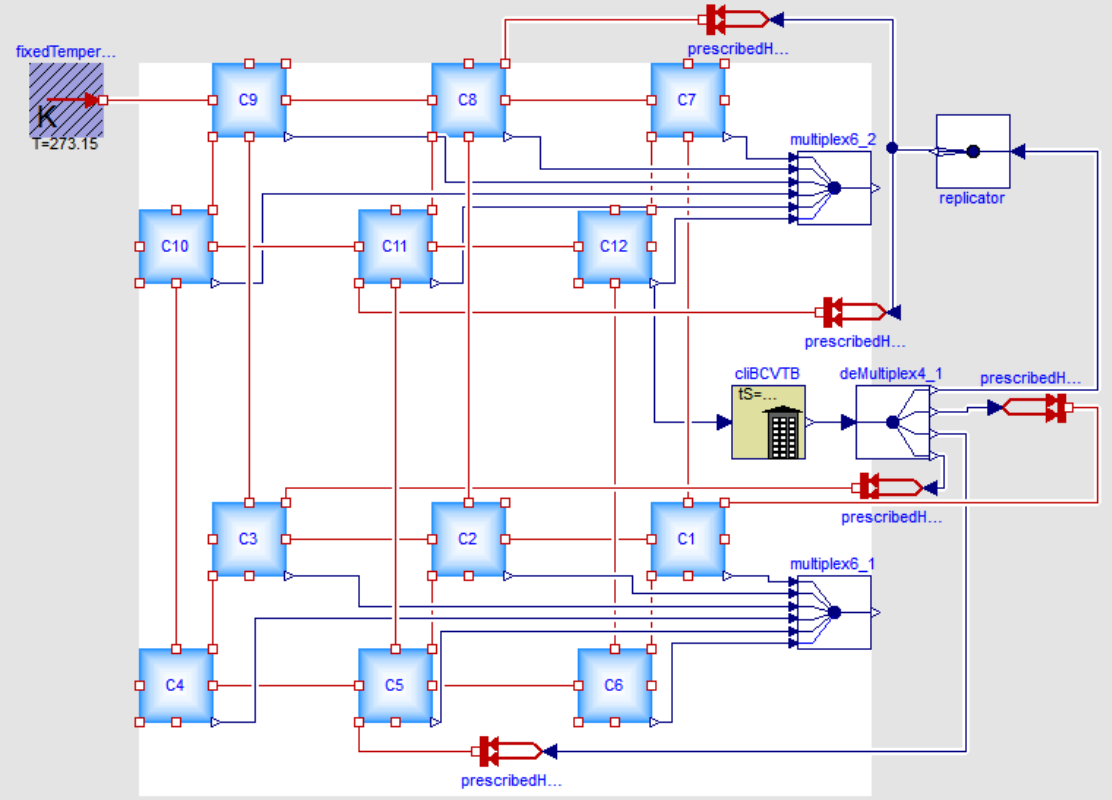

Abbildung 6.33: In Dymola/Modelica implementiertes Raummodell, verwendet in der Co-Simulation mit drei Maschinenmodellen und einem Temperaturregler

Um eine unterschiedliche Wärmeabgabe der beiden Maschinen zu erhalten, wurden für das Dymolamodell andere Werte gewählt: $R = 310\Omega$, $L = 16 \cdot 10^{-3}H$ und $k = 0.9Nm/A$. Für die Dauer des Hochfahrens dieses Motors wurden 10 Minuten gewählt, das Herunterfahren nimmt 5 Minuten in Anspruch. Der Output dieser Maschine wird über die Co-Simulation als vorgegebener Wärmefluss an Compartment 1 übergeben, wo dieser Motor seinen Platz hat.
Wie bereits in Kapitel 5.7 erwähnt, wurde die Wärmeleistung einer realen Maschine gemessen und in einer Excel-Datei gespeichert. Durch den Zugriff darauf über MATLAB wurde diese Wärme mithilfe der Co-Simulation über BCVTB dem Raummodell übergeben, in dessen Compartment 3 diese Maschine steht.
Um die Raumtemperatur trotz der abgegebenen Wärme der Maschinen in einem angenehmen Intervall für etwaige Arbeiter in der Halle zu halten, wurde dem Gesamtmodell eine Temperaturregelung in Simulink hinzugefügt. Die Beschreibung dieses Teilmodells kann in Kapitel 5.8 nachgelesen werden. Die obere Grenze für die Aktivierung der Kühlung wurde auf 21^oC gesetzt. Beim Überschreiten dieses Wertes werden vom System -200 Watt an die Compartments 8 und 11 übergeben, die sich einander gegenüberliegend im oberen

Bereich der Halle befinden. Im Fall des erneuten Unterschreitens der Soll-Temperatur von 20^oC stoppt die Regelung die Kühlung. Falls die Raumtemperatur unter 19^oC sinkt, was aufgrund einer kalten Fläche in Compartment 9 während des Stillstands der Maschinen passieren kann, werden 150 Watt zurückgegeben, um den Raum zu heizen. Die Rückgabe dieses Wertes stoppt, sobald wieder 20^oC erreicht sind. Der Wert der Raumtemperatur für den Vergleich mit dem Sollwert wird in Compartment 12 gemessen, was als Position des Thermometers der Halle interpretiert werden kann.
Ein besonderer Aspekt bei der Co-Simulation mit dieser Regelung ist die Rückkopplung, die für den Vergleich benötigt wird, sodass das Heizungs- und Kühlsystem auf die Raumtemperatur reagieren kann, was diese wiederum verändert und zu einer Änderung des Regelungsoutputs führt. Um in diesem Kreislauf keine algebraische Schleife zu erhalten, wurde dem Simulink-Modell ein Memory-Block hinzugefügt, der den Wert der Raumtemperatur aus dem letzten Synchronisationsschritt speichert und im nächsten Schritt zur Verwendung weitergibt. Die daraus resultierende verzögerte Reaktion der Temperaturregelung muss demnach in Kauf genommen werden.
Neben den zur eigentlichen Simulation notwendigen Modellen wurde über eine eigene MATLAB-Funktion die CPU-Zeit in einer Excel-Datei aufgezeichnet. Die Dauer der zusätzlichen Kommunikation und der Ausführung dieses Skripts ist derart minimal, dass die Auswirkungen auf die gesamte Simulationszeit vernachlässigt werden können. Interessant ist dabei, dass der Verlauf der Rechenzeit beobachtet werden kann, was Rückschlüsse auf die für einzelne unterschiedliche Simulationsabschnitte benötigte CPU-Zeit erlaubt und etwaige Veränderungen der Effizienz im Laufe der Simulation aufzeigt.
Mit der eben beschriebenen MATLAB-Datei zur Aufzeichnung der CPU-Zeit werden in diesem Experiment insgesamt vier MATLAB-Instanzen (zwei davon für den Aufruf von Simulink und Simscape) und zwei Dymola-Instanzen über BCVTB co-simuliert, was hinsichtlich der Performance und der reibungslosen Synchronisation bemerkenswert ist.
Wie der zum Modell für die Gesamtsimulation gehörige Signalflussgraph aussieht, zeigt Abb. 6.34.

Auf die abgegebene Wärme der drei Maschinen kann über den Output der Simulator Actors *Dymolamotor*, *Simscape* und *matlabmachine* zugegriffen werden. Diese Werte werden dem Input Port des Simulator Actors *Dymolaroom* übergeben, da sie, wie oben beschrieben, in der Simulation des Raummodells als Eingang der vorgeschriebenen Wärmeflüsse benötigt werden. Einziger Input in den BCVTB-Block des Dymola-Raummodells und somit Output des zugehörigen Simulator Actors in BCVTB ist die in Compartment 12 gemessene Temperatur, die in weiterer Folge dem Simulator Actor *Simulink*, der die Kommunikation mit der Temperaturregelung ermöglicht, zum Vergleich mit dem Sollwert übergeben wird. Der Output des *Simulink* Simulator Actors, welcher der Wärmeleistung des Kühl- bzw. Heizsystems entspricht, dient alsdann erneut als Input für das Raummodell. Der *matlabcpu* Actor dient der Kommunikation mit der simplen MATLAB-Funktion zur Dokumentation der benötigten Rechenzeit. Die Ausgänge jedes Simulators werden weiters direkt in BCVTB in einem Plotter zur raschen Veranschaulichung nach der Zeit aufgezeichnet.

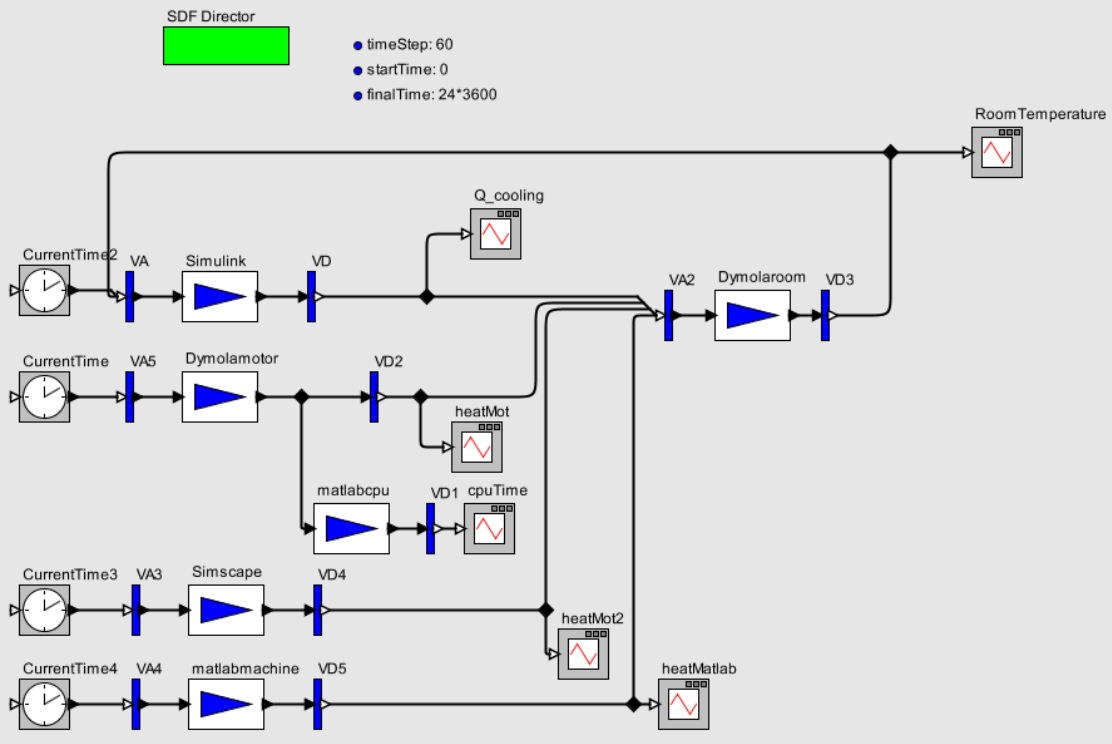

Abbildung 6.34: BCVTB-Modell zur Co-Simulation von einem Raummodell, drei Maschinenmodellen und einem Temperaturregler

Simulation des Gesamtmodells der Maschinenhalle über 24 Stunden

Im ersten Experiment mit dem eben beschriebenen Modell wurde über die Dauer von einem Werktag simuliert. Abb. 6.35 zeigt die abgegebene Wärme der Motoren, wie sie von BCVTB aufgezeichnet wurde.

Abb. 6.36 zeigt den Verlauf der Temperatur in Compartment 12 sowie den Output der Regelung, die auf diesen Temperaturwert reagiert. Es ist deutlich zu erkennen, dass der Rückgabewert der Regelung auf 150 springt, sobald die Raumtemperatur unter 19^oC sinkt. Auch die Reaktion auf die Erhöhung der Temperatur erfolgt wie vorgesehen. Die zunächst häufig erscheinenden Wechsel des Ausgangs der Regelung haben eine minimale Breite von 11 Minuten, was für das Kühlsystem eines Raumes dieser Größe ein durchaus realistisches Ergebnis ist. Weiters kann man erkennen, dass die Kühlung einige Zeit benötigt, um den raschen Anstieg der Temperatur durch das Hochfahren und die damit verbundene Abgabe von Wärme zu drücken. Insgesamt bleibt die Temperatur jedoch zwischen 18.7 und 22.2^oC, was etwa 291.8 und 295.3 Kelvin (siehe Skala) entspricht. Diese Schwankung liegt in einem durchaus tolerablen Bereich für Menschen, die die Halle betreten könnten.

Bei der Betrachtung des Temperaturverlaufs in den unteren sechs Compartments (Abb. 6.37 kann beobachtet werden, dass die Temperatur in den Compartments, in welchen sich Maschinen befinden (dargestellt in den Farben blau, schwarz und grün), erwartungsgemäß

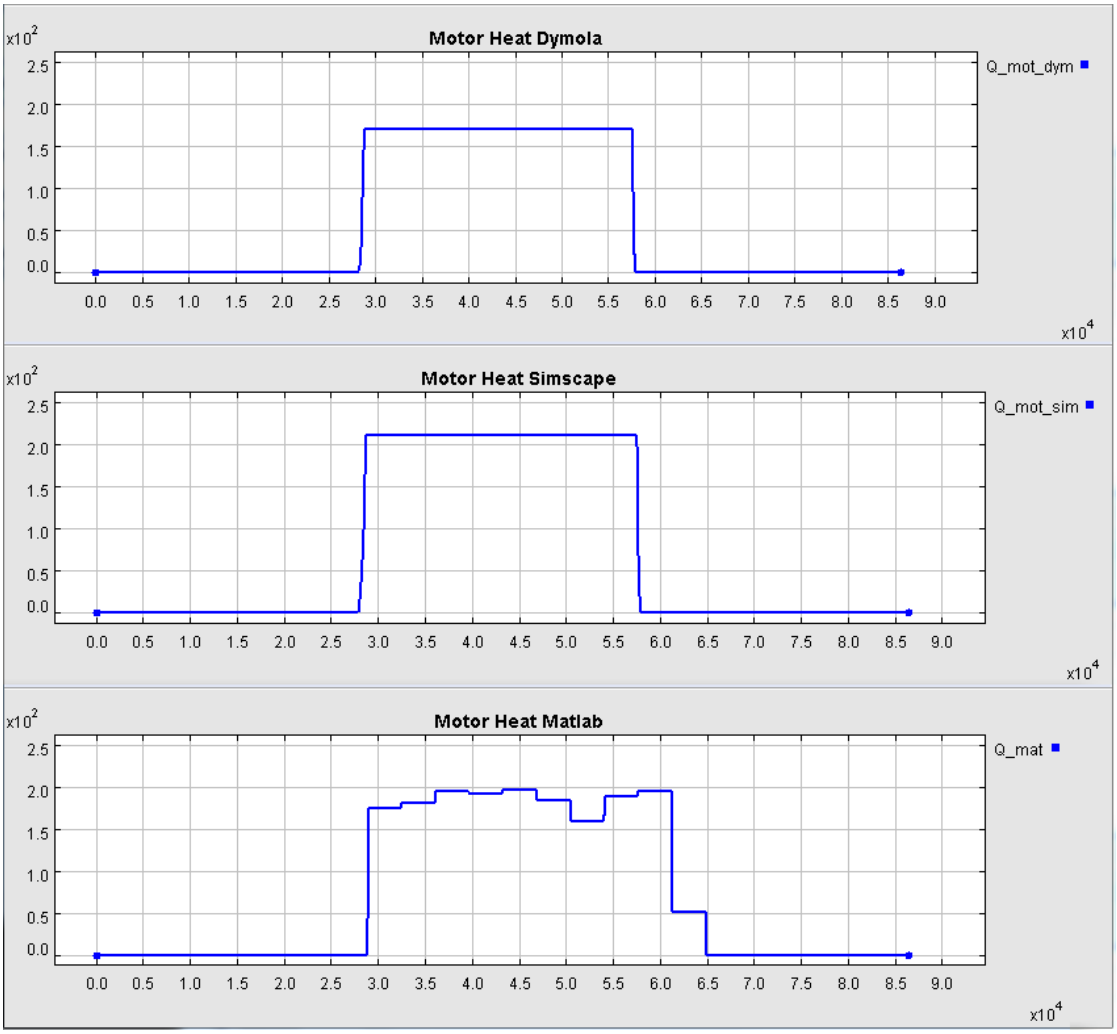

Abbildung 6.35: Verlauf der abgegebenen Wärme der verschiedenen Maschinenmodelle

schneller auf deren abgegebene Wärme reagiert.

Der Verlauf der Temperaturen in den oberen Compartments der Halle ist vor allem von den dort befindlichen Ausgängen des Regelungssystems geprägt, wie Abb. 6.38 zeigt. Wieder kann beobachtet werden, dass die Compartments, in denen die Eingänge münden (rote und schwarze Linie), schneller auf deren Veränderungen reagieren. Die Ähnlichkeit des Verlaufs in direkt übereinander liegenden Compartments ist weniger ausgeprägt als bei seitlich benachbarten, da die Konvektionskoeffizienten für den Austausch an horizontalen Flächen generell niedriger ausfallen als bei vertikalen.

Die Zeitschritte, die die individuellen Solver der Simulatoren zwischen zwei Synchronisationszeitpunkten setzen, sind in Abb. 6.39 dargestellt.

Die schwarze Linie zeigt die fixen Zeitschritte von 60 Sekunden des SDF-Directors zur

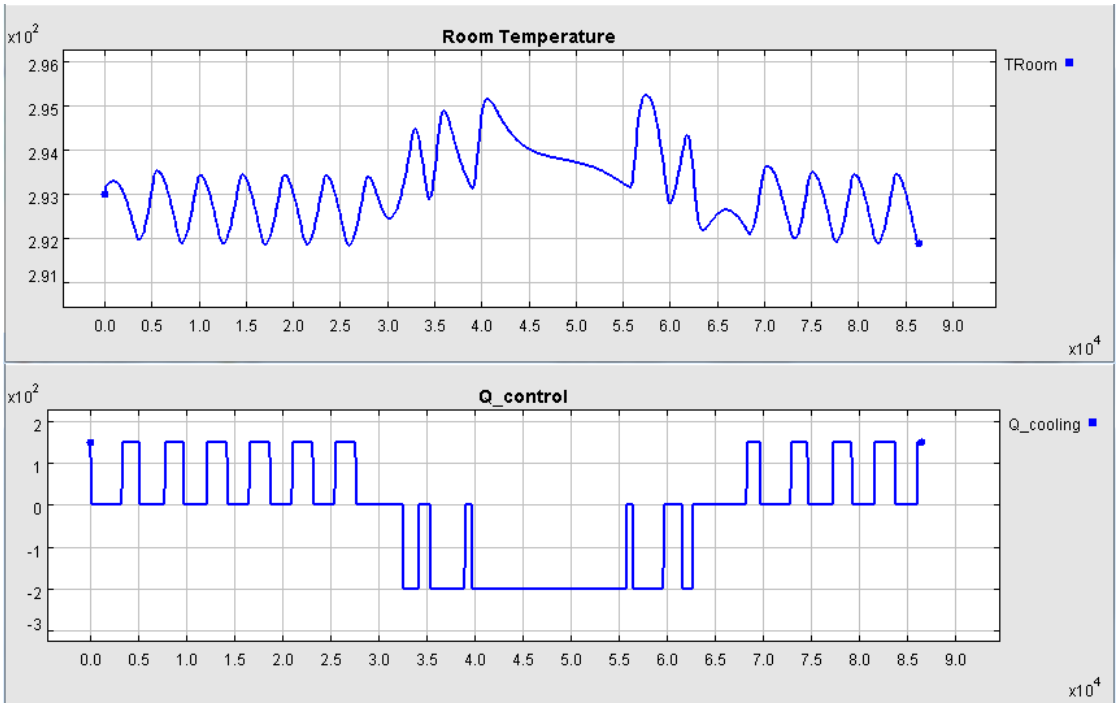

Abbildung 6.36: Wechselwirkung zwischen der gemessenen Raumtemperatur und dem Output des Heiz- und Kühlsystems

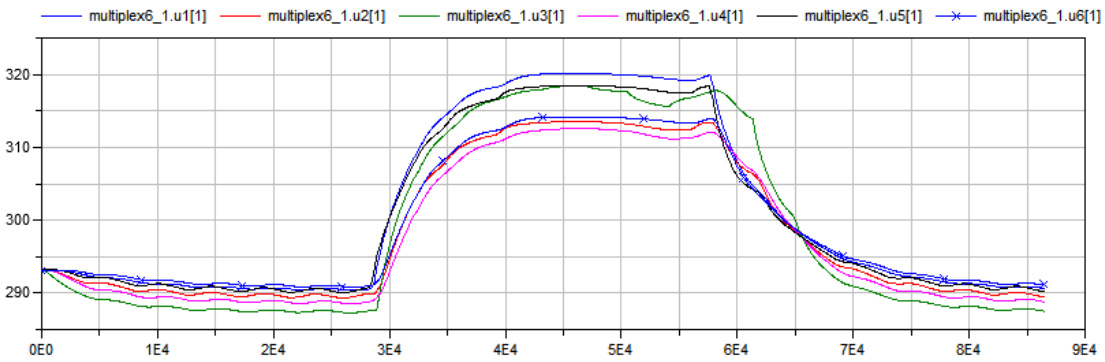

Abbildung 6.37: Temperaturverlauf in den unteren sechs Compartments in der Simulation über einen Tag

Gesamtsimulation über BCVTB. Für die Simulation der Temperaturregelung, in der nur diskrete Zustände angenommen werden, wurde der Solver *discrete* gewählt. Da dieses System lediglich zu den Synchronisationszeiten von einer Änderung der Raumtemperatur erfahren kann und sich somit nur zu diesen Zeitpunkten die Zustände des Regelungssystems verändern, entsprechen die Zeitschritte für diese Teilsimulation (dargestellt in hellblau) exakt den

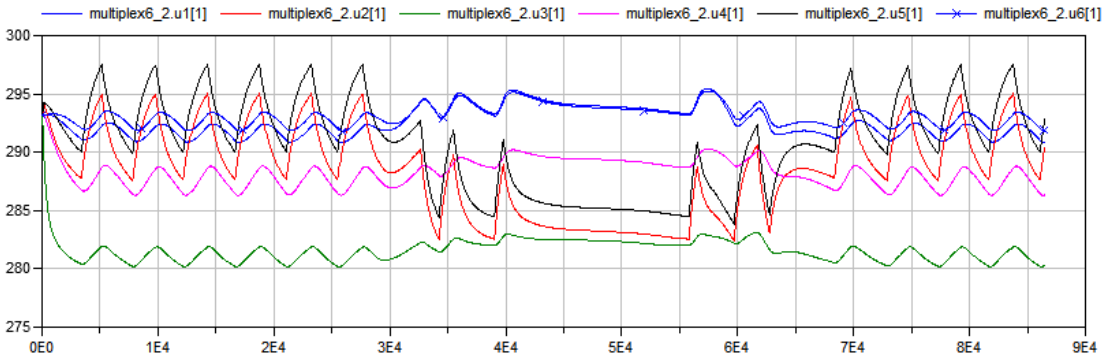

Abbildung 6.38: Temperaturverlauf in den oberen sechs Compartments in der Simulation über einen Tag

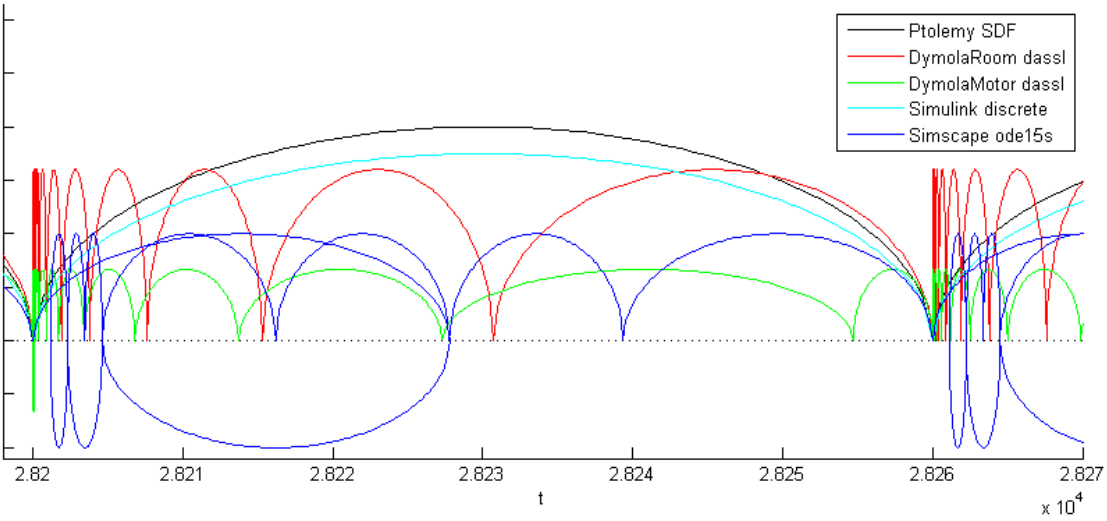

Abbildung 6.39: Zeitschritte der einzelnen Solver zwischen zwei Synchronisationszeitpunkten

Zeitschritten des SDF-Directors.
Die Schritte, die der Dassl-Solver für die Simulation des Raumes setzt, entsprechen der roten Linie. Nach jeder Synchronisation werden zunächst kleine Schritte benötigt, um die sprungartigen Veränderungen der über BCVTB erhaltenen Werte zu verarbeiten. Nach einigen Sekunden des Einpendelns kann man eine durchschnittlich etwas größere Schrittweite für den Raum als für die beiden Motoren beobachten. An den Sprüngen in der unteren Halbebene ist zu erkennen, dass sowohl der Dassl-Solver von Dymola (grüne Linie) als auch der für die Simulation des Simscape-Modells gewählte Solver *ode15s* (dunkelblau) nach der Synchronisation Schritte verwerfen. Der Synchronisationszeitpunkt wird jedoch von jedem Solver ohne Iteration exakt erreicht.

Zur Co-Simulation des Gesamtmodells über einen Tag werden insgesamt lediglich 66.7 Sekunden Rechenzeit benötigt, was angesichts der vielen verschiedenen beteiligten Simulatoren ein bemerkenswertes Ergebnis ist.

Simulation der Maschinenhalle über sieben Tage

In Abb. 6.40 und 6.41 wird der Temperaturverlauf in den Compartments bei der Simulation des gleichen Modells über sieben Tage gezeigt. Durch den Stillstand der Maschinen in den Nachtstunden und die Regelung der Temperatur liegt diese zu Beginn des zweiten Tages wieder im Bereich der ursprünglichen Temperatur, wodurch sich als Wochenverlauf das Bild einer weitgehend periodischen Wiederholung der Ergebnisse aus der Simulation über einen Tag ergibt.

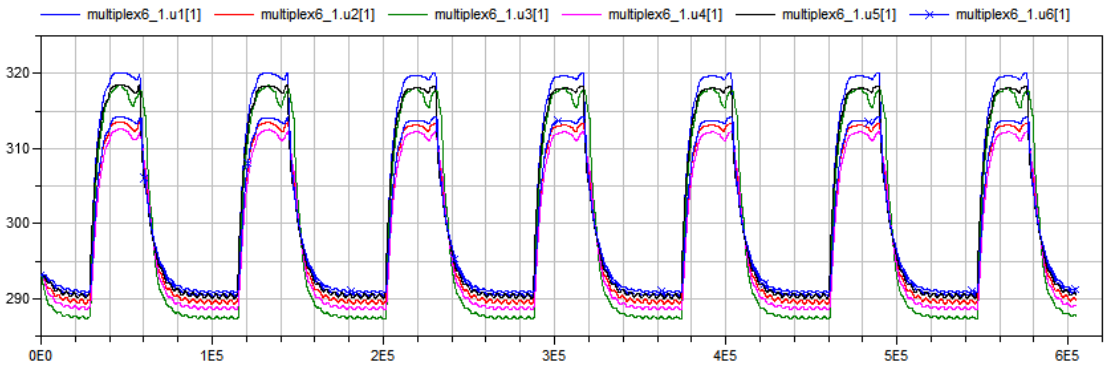

Abbildung 6.40: Temperaturverlauf in den unteren sechs Compartments bei der Simulation über sieben Tage

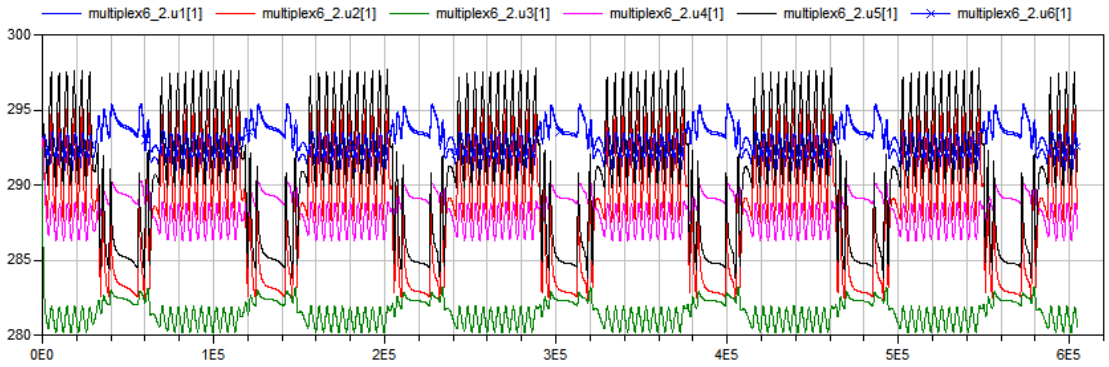

Abbildung 6.41: Temperaturverlauf in den oberen sechs Compartments bei der Simulation über sieben Tage

Bei der Beobachtung der verbrauchten Rechenzeit kann man feststellen, dass die Simulation weder langsamer vor sich geht, je länger sie bereits dauert, noch ab irgendeinem

Zeitpunkt schneller verläuft. Insgesamt kann ein weitgehend linearer Verlauf festgestellt werden, was auch die benötigte Gesamt-Rechenzeit von 469 Sekunden bekräftigt.

Simulation über ein Jahr

Um zu testen, wie das Co-Simulationstool BCVTB mit Langzeitsimulationen umgeht, wurde das Modell in einem weiteren Experiment über ein ganzes Jahr simuliert, was auch dem Ziel des INFO-Projekts mit dessen endgültigem Modell entspricht.
Die Simulation verlief auf Anhieb ohne jegliche Fehlermeldung oder gar einem daraus resultierenden vorzeitigen Abbruch durch einen der Simulatoren. Die benötigte Rechenzeit stieg auch in der Simulation über diesen überaus langen Zeitraum im Laufe des Experiments nie schneller oder langsamer an, sondern zeigte einen gleichmäßigen, linearen Verlauf und entsprach an einem gewöhnlichen Stand-PC mit Quad-Core Prozessor 6 Stunden und 40 Minuten.

6.1.1.7 Simulation eines Gebäudemodells unter Berücksichtigung von Wandeigenschaften und Außenbedingungen

Während in den bisherigen Experimenten die Temperaturverteilung innerhalb eines Raumes, für den von einer kompletten Isolation nach außen ausgegangen wurde, im Vordergrund stand, wird für die folgende Simulation ein Modell für ein Gebäude, bestehend aus drei Räumen, verwendet (siehe Abb. 6.42). Das Innere jedes dieser Räume wird durch ein thermisches Luftcompartment dargestellt, dessen Wände durch zwei Festkörpercompartments modelliert sind, die die Eigenschaften einer Gips- bzw. Beton- und einer Isolationsschicht repräsentieren. Mit dem Compartment für die äußerste Wandschicht wird eine konstante Temperaturquelle verbunden, die die Außenbedingungen des Gebäudes verkörpern soll. An der Ostseite des Compartments Nr. 3 ist ein Fenster angebracht, das aus einer einfachen Glasschicht, ebenfalls über ein Festkörpercompartment realisiert, besteht. Die drei Räume sind durch einfache Gipswände getrennt.
Compartment Nr. 1 wird die Abwärme der in Simscape und Dymola implementierten Maschinen über den BCVTB-Block hinzugefügt.

Dass Zusammenspiel, das in der Co-Simulation mit BCVTB zwischen den einzelnen Simulatoren stattfinden soll, zeigt Abb. 6.43.

Es wird davon ausgegangen, dass zu Beginn der Simulation alle Compartments eine Temperatur von 15^oC aufweisen. Wie Abb. 6.44 zeigt, ändern sich diese Temperaturen erwartungsgemäß aufgrund der gleichen Außentemperatur bis zum Einschalten der Maschinen nicht. Compartment 1, in dem sich die Maschinen befinden, reagiert am schnellsten auf deren Wärmeabgabe, jedoch sind auch in den baulich getrennten Compartments 2 und 3 leichte Reaktionen ersichtlich. Da eine Wand von Compartment 3 aus einem verhältnismäßig schlechter isolierten Fenster besteht, wird es nicht nur wegen der weiten Entfernung zu dem Compartment mit den Maschinen weniger erwärmt; die Abkühlung erfolgt auch deutlich schneller. Insgesamt ist in allen Compartments zu beobachten, dass sich diese lediglich um wenige Kelvin erwärmen und auch eine komplette Abkühlung innerhalb weniger Stunden nach dem Abschalten der Maschinen ohne eingebaute Regelung vonstattengeht. Dies ist

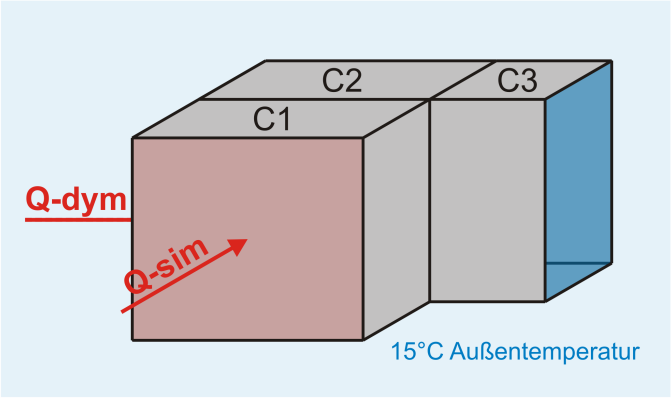

Abbildung 6.42: Skizze eines Gebäudemodells mit drei Räumen

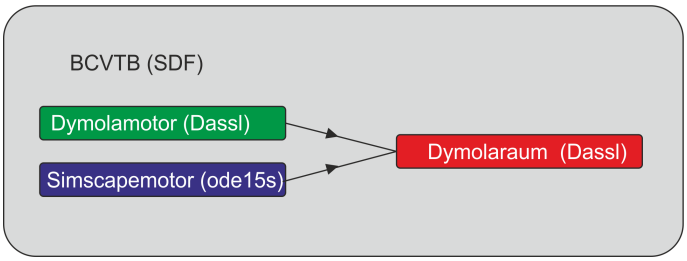

Abbildung 6.43: Vorgesehenes Zusammenspiel der beteiligten Simulatoren in der Co-Simulation des Gesamtmodells

dadurch zu erklären, dass das Gebäude zur 15^oC kühlen Außenwelt nicht total isoliert ist, sondern die Wandmaterialen eine gewisse Wärmeleitfähigkeit aufweisen. Insgesamt wurde das Modell für die Dauer von sieben Tagen simuliert, in denen durch die Möglichkeit zur Abkühlung während der Nacht- und somit Ruhestunden der Maschinen die Raumtemperatur insgesamt einen annähernd periodischen Verlauf zeigt.

Das Dymola-Gebäudemodell besteht aufgrund der vielen Schichten der Wände aus insgesamt 32 Compartments, was alle bisher vorgestellten Raummodelle an Komplexität weit übersteigt. Beachtenswert ist, dass diese Tatsache kaum Auswirkungen auf die Zeitschritte des Dassl-Solvers für das Raummodell hat, was unter anderem beim Vergleich der Schritte in Abb. 6.45 mit jenen bei einem einfacheren Modell, siehe etwa Abb. 6.17, festgestellt werden kann.

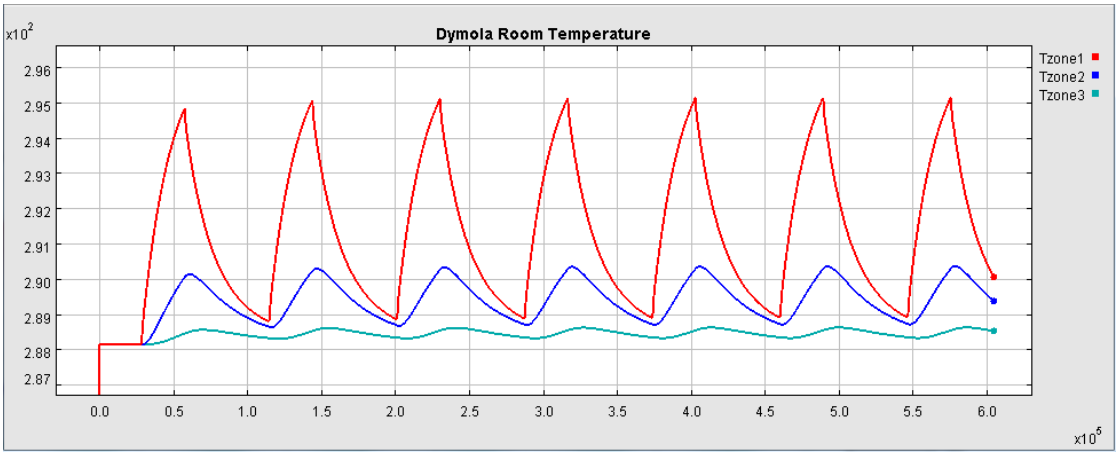

Abbildung 6.44: Verlauf der Temperatur in den drei Räumen über eine Woche

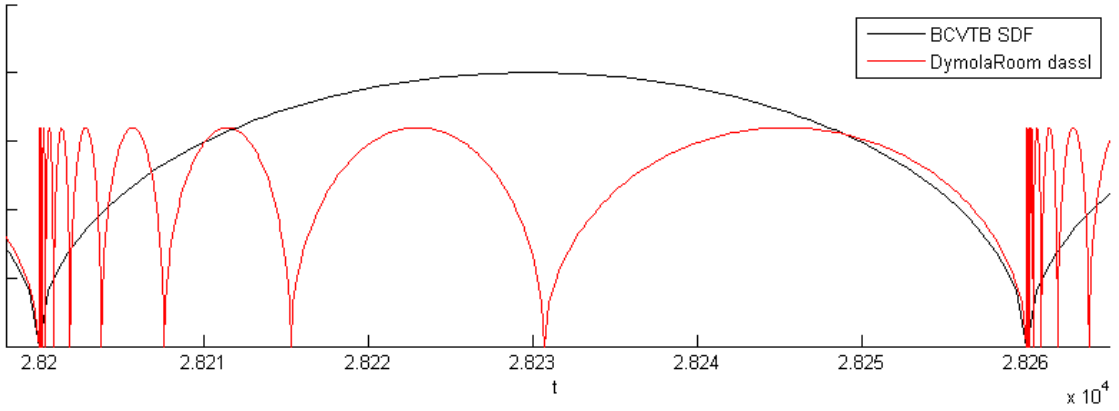

Abbildung 6.45: Zeitschritte des Dassl-Solvers für das Raummodell in einem Synchronisationsintervall (60s)

6.1.2 Simulation unter Verwendung eines in EnergyPlus implementierten Raummodells

In der Endfassung des in Kapitel 1.1 vorgestellten INFO-Projekts soll eine Maschinenhalle über das Gebäudesimulationstool EnergyPlus, das in Kapitel 4.5 beschrieben wird, modelliert werden. Aus diesem Grund wird in diesem Abschnitt die Co-Simulation eines in dieser Software erstellten Raummodells mit den bereits vorgestellten Maschinen, implementiert in Modelica bzw. Simscape, berücksichtigt. Um zudem eine Möglichkeit zu erhalten, die Simulatoren Dymola und EnergyPlus für Gebäudemodelle zu vergleichen, wurde für dieses Experiment das im vorhergehenden Kapitel vorgestellte Raummodell in EnergyPlus nachgebaut.
Ein Überblick der in dieser Simulation verwendeten Simulatoren und deren Austausch ist in

Abb. 6.46 gegeben.

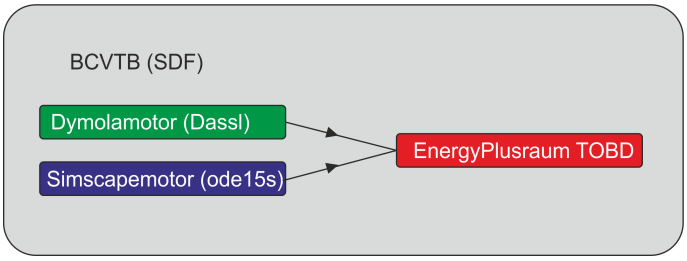

Abbildung 6.46: Vorgesehene Kommunikation zwischen den Simulatoren EnergyPlus, Dymola und Simscape über BCVTB

Da die Co-Simulation mit EnergyPlus nur mit einer Wetterdatei erfolgen kann, wurde versucht, eine konstante Wetterdatei zu erstellen, um dieselben Außenbedingungen, die für das in Dymola implementierte Gebäudemodell verwendet wurden, zu schaffen. In einer Wetterdatei kann jedoch nicht dezidiert die Außentemperatur angegeben werden, sondern nur eine so genannte *dry-bulb temperature*, aus der EnergyPlusintern die Luft- und Oberflächentemperatur berechnet wird. Experimentell wurde ermittelt, dass bei einer *dry-bulb temperature* von 22^oC die Temperaturen in den drei Zonen ohne weitere Zufuhr oder Wegnahme von Wärme konstant bei ca. 14.3, 14.4 bzw. 15.6^oC bleiben, was für die Vergleichbarkeit der beiden Simulationen als hinreichend nahe bei 15^oC betrachtet wurde. Der Unterschied in den Zonentemperaturen selbst kann dadurch erklärt werden, dass Zone 3, in der sich das Fenster befindet, stärker durch die höheren Außentemperaturen beeinflusst wird, was die etwas höhere Zonentemperatur im Vergleich zu den beiden anderen ergibt.
Die Zonentemperaturen nach der Simulation dieses Modells über sieben Tage sind in Abb. 6.47 gezeigt.

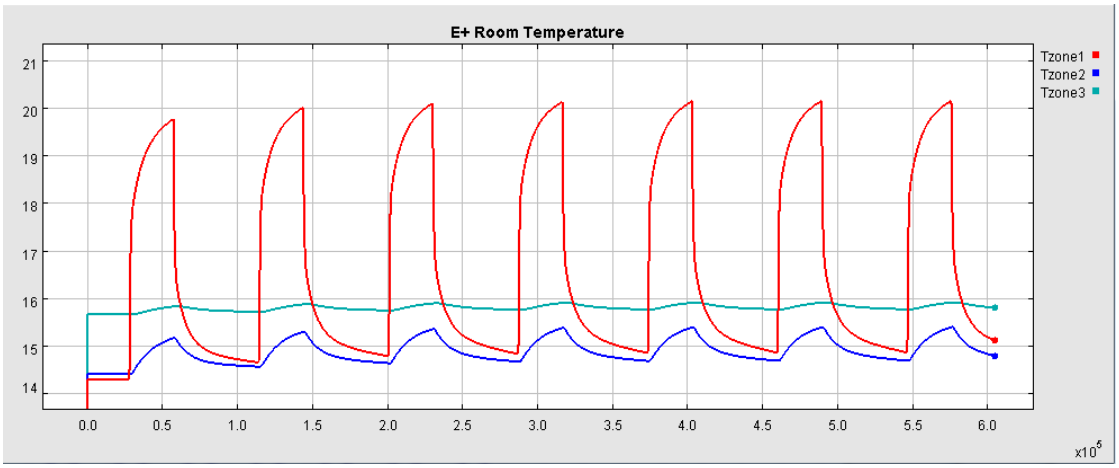

Abbildung 6.47: Verlauf der Temperatur in den drei Räumen über eine Woche

Wie zu erwarten war, reagiert Zone 1, in der sich die Maschinen befinden, am schnellsten und intensivsten auf deren Wärmeemission, während die entfernteste und zur Außenwelt am wenigsten isolierte Zone 3 nur kleine Veränderungen zeigt.
Für die Zeitschritte der Solver für die Maschinenmodelle ist zu erwarten, dass sie keine Veränderung zu vorhergehenden Simulationen zeigen, da diese durch keine Werte der anderen beteiligten Simulatoren beeinflusst werden. Abbildung 6.48 bestätigt diese Annahme.

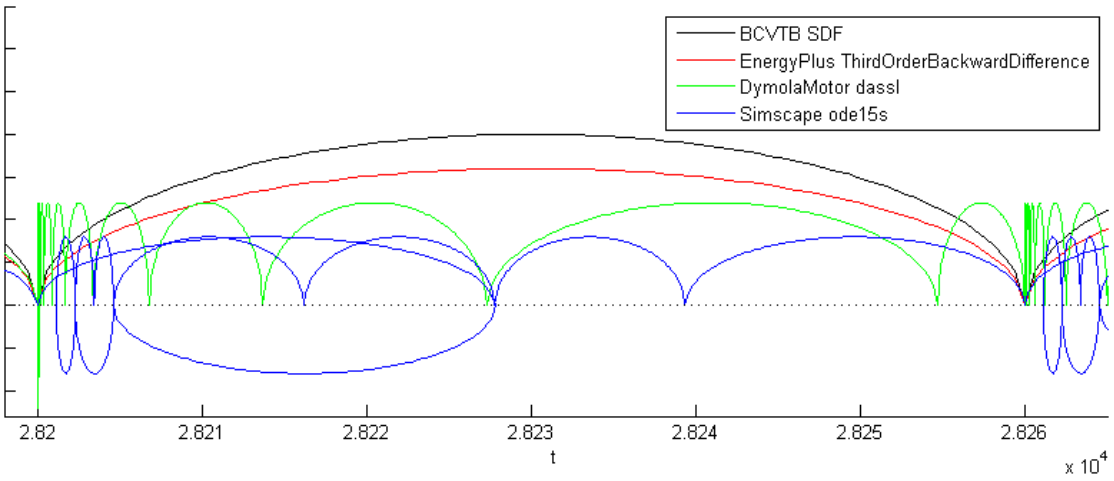

Abbildung 6.48: Zeitschritte der Solver für die Simulation des Gesamtmodells mit EnergyPlus-Gebäudemodell in einem Synchronisationsintervall (60s)

An dieser Stelle sei erwähnt, dass für EnergyPlus nur der Zeitschritt bekannt ist, der in den Simulationseinstellungen angegeben wurde. Falls aus Konvergenzgründen dennoch kleinere Schritte verwendet wurden, so erhält man als Benutzer keinerlei Information darüber. Der in Abb. 6.48 dargestellte Zeitschritt entspricht demnach nur dem gewählten Wert, da die tatsächlich intern verwendeten Schritte nicht berücksichtigt werden können.

6.1.3 Vergleich der Gebäudemodelle und deren Simulation in Dymola und EnergyPlus

Einer der ersten Unterschiede, der beim Vergleich der beiden Modellbildungsarten, wie sie in Dymola bzw. EnergyPlus verwendet werden, auffällt, ist die Prioritätensetzung. Während bei der Entwicklung der Compartments in Modelica/Dymola die Bestimmung der Temperaturverteilung in einem Raum im Vordergrund stand, ist es in EnergyPlus nicht vorgesehen, einen Raum weiter zu unterteilen. Im Gegenteil werden in EnergyPlus-Modellen eher mehrere Räume zu einer Zone zusammengefasst, falls die Unterschiede in deren thermischem Verhalten vernachlässigt werden können. Im Hinblick auf äußere Bedingungen hingegen können in EnergyPlus deutlich mehr Einflüsse berücksichtigt werden, wie die Ausrichtung des Gebäudes nach Himmelsrichtungen, eigene Beschattungselemente als auch detaillierte Wetterdaten aus den eigens für EnergyPlus generierten .epw-Dateien, die neben der *dry-bulb temperature* auch Sonneneinstrahlungswinkel, Bewölkungsgrad und Luftdruck je-

des Tages enthalten. Andererseits werden aus diesen Daten derart viele Werte weiterverarbeitet, über die der Benutzer keinen Überblick mehr hat, sodass viele Ergebnisse nicht nachvollziehbar sind. Die äußerst ausführliche Dokumentation, die in mehrere Einzeldokumente von jeweils bis zu knapp 2000 Seiten unterteilt ist, dient hierbei nur bedingt guter Auskunft, da es viel Zeit und Ausdauer bedarf, eine Suche nach einem konkreten Problem erfolgreich abzuschließen, und dadurch bei Weitem noch kein Überblick über die Gesamtverarbeitung gegeben ist. Im Gegensatz dazu sind die unter Dymola verwendeten Werte und Simulationsvorgänge alle bekannt, was auch dadurch zu begründen ist, dass beim Anwender von Modelica bzw. Dymola gewisse Programmierkenntnisse vorausgesetzt werden und dieser auch vollen Zugriff auf den dahinterliegenden Code hat, was den Benutzern von EnergyPlus, die oftmals aus dem Bereich der Gebäudetechnik kommen und mit der hinter der Simulation stehenden Mathematik wenig bewandt sind, vorenthalten bleibt. Somit kann zusammenfassend festgestellt werden, dass die Absichten der beiden Simulationszwerkzeuge sich grundsätzlich unterscheiden, was einen detaillierten Vergleich nicht zweckmäßig macht.
Hinsichtlich der Ergebnisse aus den Simulationen der Kapitel 6.1.1.7 und 6.1.2 kann bemerkt werden, dass die Reaktion der Raumtemperaturen in Dymola etwas langsamer erfolgt. Dies kann dadurch erklärt werden, dass innerhalb der Zone dennoch Positionen unterschieden werden, so befinden sich die beiden Maschinen im Dymolamodell an zwei Außenwänden des Compartments 1; die von ihnen abgegebene Wärme muss, salopp ausgedrückt, somit erst den Weg zum anderen Ende des Compartments zurücklegen, bis von dort eine Weiterleitung an die Wände und weiter an die anderen Compartments erfolgt. Außerdem wird die Temperatur in den Compartments immer in deren Mitte gemessen, so kommt es auch im Compartment mit den Maschinen selbst zu einer verzögerten Reaktion. In EnergyPlus hingegen wird die Wärmequelle schlichtweg der Zone ohne weitere Positionierung zugeordnet, was vermuten lässt, dass die Auswirkungen auf die Temperatur und der Austausch mit den Wandflächen ohne Verzögerung stattfindet.
Die resultierenden Temperaturkurven sind jedoch im Blick auf die Größenordnung und den allgemeinen Verlauf durchaus vergleichbar, auch die Erwärmung der Compartments bzw. Zonen ohne direkte Wärmequellen zeigt sich in einander grundsätzlich entsprechenden Abständen zum Maschinen beinhaltenden Compartment.
Hinsichtlich der verbrauchten Rechenzeit der beiden Simulationen ist zu bemerken, dass EnergyPlus trotz vieler unbemerkt im Hintergrund stattfindender Operationen mit 33 Sekunden für eine Woche Simulationszeit deutlich sparsamer als Dymola mit 139 Sekunden für dieselbe Simulationsdauer arbeitet. Bei näherer Betrachtungsweise ist dies jedoch nicht weiter überraschend, da EnergyPlus - soweit es dem Benutzer bekannt ist - mit einer Schrittweite von 60 Sekunden Berechnungen durchführt, während vom Solver in Dymola beinahe das Zehnfache dieser Schritte in derselben Simulationszeit gesetzt wird.
Abschließend kann festgehalten werden, dass beide Simulatoren andere Schwerpunkte setzen, sich von mathematischer Sichtweise jedoch Dymola als durchsichtiger und somit gezielter bedienbar erweist.

6.2 Grenzen von BCVTB

Eines der wesentlichen Ziele dieser Arbeit war es, aufzuzeigen, welche Möglichkeiten der Co-Simulation mit dem verwendeten Tool BCVTB bestehen und wo dessen Grenzen liegen. Einige erste Stolpersteine legten sich bereits der Inbetriebnahme dessen in den Weg. So war es zur Ermöglichung der Einbindung von MATLAB beispielsweise notwendig, die 32-bit Version zu installieren, da die 64-bit Version mit Version 1.0.0 von BCVTB nicht kompatibel ist, was zu per se nicht darauf hinweisenden Fehlermeldungen bei der versuchten Kopplung führt. Ab Version 1.1.0 wurde dieses Problem laut Angaben der Entwickler bereinigt. Ein weiterer Umstand, der zur erfolgreichen Durchführung der Co-Simulation von Dymolamodellen erst lokalisiert und behoben werden musste, war die Löschung von Kompilationsdateien von Dymola, die ursprünglich nicht erfolgte, sobald eine Simulation durchgeführt worden war, was jede Änderung für weiteren Simulationen verhinderte. Diesem Problem wurde durch die Modifikation der von BCVTB mitgelieferten Datei rundymola.bat, sodass diese Dateien immer gelöscht werden, sobald sie existieren, Abhilfe geschafft.
Nach der erfolgreichen Installation aller beteiligten Umgebungen in kompatiblen Versionen und den in Kapitel 5.1 beschriebenen Veränderungen und Zusätzen zur Ermöglichung der Synchronisation konnten tatsächlich erste Beispiele co-simuliert werden. Im Anschluss an diese Vorbereitungen war es ohne weitere Schwierigkeiten möglich, unterschiedliche Simulatoren und auch mehrere Instanzen derer zu koppeln, solange immer eine fixe Kommunikationsschrittweite verwendet und die Co-Simulation durch einen SDF Director gesteuert wurde. Die Kommunikation zu variablen Zeitpunkten, die beispielsweise in Dymola und EnergyPlus mit den vorhandenen Komponenten ohnehin nicht möglich wäre, kann auch mit Simulatoren, die den Kommunikationszeitpunkt iterieren können, bei der Verwendung von BCVTB nicht erfolgen.
Maximal wurden im Zuge dieser Arbeit vier MATLAB- und zwei Dymolainstanzen über BCVTB gekoppelt. Die Anzahl an beteiligten Simulatoren stellte keineswegs ein Problem dar; vielmehr erwies es sich als relativ unkompliziert, viele Teilmodelle zu koppeln, sofern es schon einmal für wenige gelungen war.
Hinsichtlich der Komplexität der Einzelmodelle kann ebenfalls festgestellt werden, dass keinerlei Inkonsistenzen in der Simulation auftauchen. Einzig ist festzuhalten, dass die Kopplung eines sehr aufwändigen Teilmodells, das selbst schon ungeheure Rechenzeit beansprucht, das Gesamtsystem auch dementsprechend verlangsamt. Man muss demnach abwägen, ob für Langzeitsimulationen eine zielgerichtete Vereinfachung sehr komplexer Einzelmodelle nicht zweckdienlicher wäre.
Wie das Beispiel der Co-Simulation zweier Dymolamodelle in 6.1.1.2 zeigt, kann BCVTB auch dazu verwendet werden, einzelne steife Teile eines Systems zu entkoppeln, um sie mit einer Co-Simulation so zusammenzuführen, dass jedes Teilmodell mit einer individuellen Schrittweite arbeiten kann.
Im Hinblick auf die Dauer der angestrebten Co-Simulation haben einige Experimente über Simulationszeiträume bis zu einem Jahr (siehe etwa Kapitel 6.1.1.6) gezeigt, dass dies problemlos mithilfe von BCVTB möglich ist, solange durch die benötigte Rechenzeit der Teilmodelle die Dauer der Gesamtsimulation in einem vernünftigen Rahmen gehalten werden

kann. Diese ist durch ihre annähernd lineare Fortpflanzung relativ gut abschätzbar.
Grundsätzlich gäbe es in der Co-Simulationsumgebung BCVTB auch die Möglichkeit, Modelle zu verschachteln, in denen jedes mit einem eigenen Director simuliert wird. Allerdings kann in diesem Fall der Director auf der untersten Ebene nur ein SDF-Director sein, der im Falle eines darüberstehenden SDF-Directors denselben Zeitschritt wie dieser haben muss, was die Hierarchie obsolet macht. Ein Continuous Director in der obersten Stufe wiederum kann, wie bereits erklärt, nicht für eine Co-Simulation verwendet werden, weshalb diese Schachtelung nicht weiter in Betracht gezogen wurde.
Abschließend sei erwähnt, dass hinsichtlich der Genauigkeit bzw. des Verlusts derer je nach Modell entschieden werden muss, ob dieses für eine Co-Simulation über BCVTB geeignet ist. Da die von einem anderen Simulator erhaltenen Werte in einem Synchronisationsintervall konstant gehalten werden, können sich bei dieser Art der Kopplung über eine längere Simulation immense Fehler anhäufen. Dies kann bei sich langsam ändernden thermischen Systemen toleriert werden, wird für viele andere Gebiete mit kleineren Zeitkonstanten, in denen hohe Genauigkeit gefordert wird, jedoch zu einem großen Problem, da auch der Synchronisationsschritt nur bis zu einem gewissen Grad verkleinert werden kann, ohne die verbrauchte CPU-Zeit unverhältnismäßig in die Höhe schnellen zu lassen.

KAPITEL 7

Conclusio

In dieser Arbeit wurde die Co-Simulationsmethode des *loose coupling* mit Schwerpunkt auf den Bereich des Physical Modelling erörtert. Diese Art der Co-Simulation erwies sich als gut geeignet, um komplexe Modelle hinsichtlich Steifigkeit, Modellierungs- und Simulationsmethoden in entsprechende Teilsysteme aufzuteilen und über eine Gesamtsimulation wieder zusammenzuführen. Durch die individuell angepassten Schrittweiten können Rechenschritte eingespart und somit die Effizienz der Gesamtsimulation erhöht werden. Dennoch gilt es zu berücksichtigen, dass je nach Extrapolationsmethode sowohl der Jacobi- als auch der Gauß-Seidel-Typ eine Fehlerakkumulation bewirken, die ab einer gewissen Simulationszeit signifikante Auswirkungen auf die Resultate hat. In der über das Co-Simulationstool BCVTB überwachten Simulation erfolgt eine Extrapolation mit dem Wert der letzten Synchronisation, was neben der resultierenden Ungenauigkeit Sprünge in den Eingangssignalen aus anderen Simulatoren zur Folge hat, die vom jeweiligen Solver durch eine drastische Verringerung der Schrittweite nach jedem Synchronisationszeitpunkt behandelt werden. Daher muss abgewogen werden, wie groß das Synchronisationsintervall gewählt werden darf, um dennoch zulässig genaue Ergebnisse zu erhalten, und wo dessen untere Grenze liegt, um gleichzeitig die Rechenzeit in einem adäquaten Rahmen zu halten.
Weiters war es interessant zu beobachten, dass neben simplen diskreten Systemen wie dem Datenmodell in MATLAB oder der Regelung in Simulink auch auf gewöhnlichen Differentialgleichungen basierende Modelle, wie sie von EnergyPlus erstellt und mithilfe von finiten Differenzenmethoden bzw. Predictor-Corrector-Verfahren gelöst werden, oder differential-algebraische Gleichungssysteme, die aus akausalen physikalischen Modellen resultieren und mit entsprechenden Solvern behandelt werden, über eine Co-Simulation zusammengeführt werden können.
Dies führt zu einem weiteren Schwerpunkt dieser Arbeit, dem Bereich des Physical Modelling. Während es für Maschinenmodelle, mögen sie auch verschiedene physikalische Domänen umfassen, nahe liegt, diese in objektorientierten Sprachen akausal zu modellieren, wurde dieser Ansatz in dieser Arbeit auch verwendet, um ein Compartmentmodell für die thermischen Vorgänge in einem Gebäude zu entwickeln. Die größte Schwierigkeit dies-

bezüglich bestand darin, trotz der grundsätzlich starren Compartments die Konvektion der Luft zu berücksichtigen, was durch eine dementsprechende Neuberechnung des Wärmeübergangskoeffizienten je nach Übergangsrichtung geschah. Durch diese Vorgehensweise konnte schließlich ein Modell entwickelt werden, das vergleichbare Simulationsergebnisse zu einem ähnlich aufgebauten Modell in der Gebäudesimulationssoftware EnergyPlus lieferte, obwohl es grundsätzlich einfacher gestrickt und für den Benutzer durchsichtiger aufgebaut ist. Hinsichtlich der Modellbildungs- und Simulationsarten als auch der Benutzerzielgruppe sind diese beiden Methoden jedoch von Grund auf verschieden, sodass ein Vergleich dahingehend nicht zweckmäßig ist.
Allgemein kann festgehalten werden, dass *loose coupling* über BCVTB eine praktikable Möglichkeit darstellt, Modelle aus unterschiedlichen Bereichen schnell zu verbinden, ohne auf grobe Performanceschwierigkeiten gefasst sein zu müssen, eine Kopplung von Modellen, deren Simulation schnelle Veränderungen in den Einzelsystemen berücksichtigen muss, hohen Genauigkeitsansprüchen jedoch nicht gerecht werden kann.

KAPITEL 8

Ausblick

Neben den *loose coupling* - Co-Simulationsmethoden mit konstanter Schrittweite, auf denen der Fokus dieser Arbeit lag, bestehen auch Möglichkeiten zur Co-Simulation mit Schrittweitensteuerung oder so genannte *strong coupling* Methoden, auf die in Kapitel 3 bereits kurz eingegangen wurde. Da beim *strong coupling* bei jeder Iteration zu einem Synchronisationszeitpunkt ein Datenaustausch erfolgt, können vorgegebene Genauigkeitsvorschriften gezielt eingehalten werden, was die Verwendung bei Systemen, die vor allem aufgrund ihrer unterschiedlichen Modellierungsansprüche separiert wurden, nahelegt. Im Falle ursprünglich signifikant unterschiedlicher Schrittweiten würde diese Methode durch den häufigen Austausch zu einer deutlichen Verlangsamung des Systems führen. Ein Vergleich von *strong* und *loose coupling*-Methoden wird unter anderem in [Tr8] durchgeführt, laut dieser Quelle ist auch mit dem Gebäudesimulationstool EnergyPlus *strong coupling* möglich.
Die Möglichkeiten der Co-Simulation mit BCVTB selbst sollen laut Angaben der Entwickler dahingehend erweitert werden, als Anbindungen an neue Simulationstools geschaffen werden sollen, um auch die Verwendung von Lösungsalgorithmen, die auf Finite Elemente Methoden (FEM) oder Computational Fluid Dynamics (CFD) basieren, zu ermöglichen.
Eine weitere Methode, die vordergründig der Kopplung von Modelicamodellen mit anderen Simulationsumgebungen dient, sind die *Functional Mockup Interfaces* (kurz FMIs). Der Austausch erfolgt bei dieser Kopplungsart über eine Kombination von xml-Dateien und kompiliertem C-Code. Mithilfe eines FMI können einerseits Modelle über einen externen Solver simuliert werden als auch bei Beibehaltung der Solver der Einzelmodelle tatsächliche Co-simulationen erfolgen. Nähere Informationen zu Functional Mockup Interfaces können [Assa] entnommen werden, während ein allgemeiner Überblick über verschiedene Arten der Co-Simulation aus [Tr8] gewonnen werden kann.

ANHANG A

Appendix

A.1 Source Code der für das Compartmentmodell in Modelica entwickelten Komponenten

```
block ConvCoeff
  extends Modelica.Blocks.Interfaces.SI2SO;
  encapsulated package bla
  type gna = enumeration(
        side "side",
        up "up",
        down "down");
  end bla;

  parameter bla.gna position; //= ConvCoeff1.gna.side;
  parameter Real l "case side: l=x*y/(2*(x+y)), case up,down: l=z";
  parameter Real A "area";

protected
  constant Real a2 = 0.001068015318015; //ai..coeff of regpoly of nu
  constant Real a1 = 0.305291983016989;
  constant Real a0 = -28.044777951008353;
  constant Real b2 = 4.818245480010717*10^(-7); //bi..coeff of regpoly of Pr
  constant Real b1 = -4.174138368984302*10^(-4);
  constant Real b0 = 0.789080196186105;
  constant Real c2 = -3.795312530606681*10^(-5); //ci..coeff of regpoly of
      lambda
  constant Real c1 = 0.097175184785803;
  constant Real c0 = 0.644622988011737;
  constant Real g = Modelica.Constants.g_n;

equation
  y = if (position == 1) then A*((0.825+0.387*((g*l^3/((a2*u1^2+a1*u1+a0)
      *10^(-7))^2*(1/u1)*abs(u1-u2))*(b2*u1^2+b1*u1+b0)*((1+(0.492/(b2*u1^2+
      b1*u1+b0))^(9/16))^(-16/9)))^(1/6))^2)*(c2*u1^2+c1*u1+c0)*10^(-3)/l
```

```
28        elseif (position == 2) then A*(0.766*((g*l^3/((a2*u1^2+a1*u1+a0)
              *10^(-7))^2*(1/u1)*abs(u1-u2))*(b2*u1^2+b1*u1+b0)*((1+(0.332/(b2*
              u1^2+b1*u1+b0))^(11/20))^(-20/11)))^(1/5))*(c2*u1^2+c1*u1+c0)
              *10^(-3)/l
29        else A*(0.6*((g*l^3/((a2*u1^2+a1*u1+a0)*10^(-7))^2*(1/u1)*abs(u1-u2))
              *(b2*u1^2+b1*u1+b0)*((1+(0.492/(b2*u1^2+b1*u1+b0))^(9/16))
              ^(-16/9)))^(1/5))*(c2*u1^2+c1*u1+c0)*10^(-3)/l;
30    annotation (
31        Documentation(info="
32 <HTML>
33 </HTML>
34 "), Icon(coordinateSystem(
35      preserveAspectRatio=true,
36      extent={{-100,-100},{100,100}},
37      grid={2,2}), graphics={Text(
38        extent={{-90,-60},{90,60}},
39        lineColor={160,160,164},
40        textString="alpha(T)"), Text(
41        extent={{-150,150},{150,110}},
42        textString="%name",
43        lineColor={0,0,255})}),
44      Diagram(coordinateSystem(
45      preserveAspectRatio=true,
46      extent={{-100,-100},{100,100}},
47      grid={2,2}), graphics={Rectangle(
48        extent={{-100,-100},{100,100}},
49        lineColor={0,0,255},
50        fillColor={255,255,255},
51        fillPattern=FillPattern.Solid), Text(
52        extent={{-90,-60},{90,60}},
53        lineColor={160,160,164},
54        textString="alpha(T)")}),
55      uses(Modelica(version="3.2")),
56               uses(Modelica(version="3.2")));
57 end ConvCoeff;
```

Listing A.1: Modelica-Code für den Block zur Ermittlung des Konvektionskoeffizienten

```
1 model AirHeatCapacitor
2   parameter Modelica.SIunits.Volume V "Volume of element (= x*y*z)";
3   parameter Modelica.SIunits.Temperature Tstart
4     "Initial temperature of element";
5   Modelica.SIunits.Temperature T( start=Tstart, displayUnit="degC")
6     "Temperature of element";
7   Modelica.SIunits.TemperatureSlope der_T(start=0)
8     "Time derivative of temperature (= der(T))";
9 protected
10          Real a2 = 4.106114473762172*10^(-4);
11          Real a1 = -0.2094958989540265;
12          Real a0 = 745.3518996689215;
13          Real b2 = 1.158705510176113*10^(-5);
14          Real b1 = -0.01095625607186941;
15          Real b0 = 3.407356083066301;
16          Real rho= b2*Tstart^2+b1*Tstart+b0;
```

```
17 public
18   Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port annotation (
19       Placement(transformation(extent={{-8,-104},{12,-84}}),
             iconTransformation(
20           extent={{-12,-100},{8,-80}})));
21
22 equation
23   T = port.T;
24   der_T = der(T);
25   //C*der(T)=c_v*m=c_v*V*rho, using regression polynomial for c_v(T),rho(T)
26   (a2*T^2+a1*T+a0)*V*rho*der(T) = port.Q_flow;
27    annotation (
28     Icon(coordinateSystem(preserveAspectRatio=true, extent
           ={{-100,-100},{100,100}}),
29                          graphics={
30         Text(
31           extent={{-150,110},{150,70}},
32           textString="%name",
33           lineColor={0,0,255}),
34         Polygon(
35           points={{0,67},{-20,63},{-40,57},{-52,43},{-58,35},{-68,25},{-72,
36               13},{-76,-1},{-78,-15},{-76,-31},{-76,-43},{-76,-53},{-70,-65},
37               {-64,-73},{-48,-77},{-30,-83},{-18,-83},{-2,-85},{8,-89},{22,
38               -89},{32,-87},{42,-81},{54,-75},{56,-73},{66,-61},{68,-53},{
39               70,-51},{72,-35},{76,-21},{78,-13},{78,3},{74,15},{66,25},{54,
40               33},{44,41},{36,57},{26,65},{0,67}},
41           lineColor={0,0,255},
42           fillColor={80,180,255},
43           fillPattern=FillPattern.Solid),
44         Polygon(
45           points={{-58,35},{-68,25},{-72,13},{-76,-1},{-78,-15},{-76,-31},{
46               -76,-43},{-76,-53},{-70,-65},{-64,-73},{-48,-77},{-30,-83},{-18,

47               -83},{-2,-85},{8,-89},{22,-89},{32,-87},{42,-81},{54,-75},{42,
48               -77},{40,-77},{30,-79},{20,-81},{18,-81},{10,-81},{2,-77},{-12,
49               -73},{-22,-73},{-30,-71},{-40,-65},{-50,-55},{-56,-43},{-58,-35},

50               {-58,-25},{-60,-13},{-60,-5},{-60,7},{-58,17},{-56,19},{-52,
51               27},{-48,35},{-44,45},{-40,57},{-58,35}},
52           lineColor={0,0,255},
53           fillColor={0,128,255},
54           fillPattern=FillPattern.Solid),
55         Text(
56           extent={{-69,7},{71,-24}},
57           lineColor={0,0,0},
58           textString="%C")}),
59     Diagram(coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},
60             {100,100}}), graphics={
61         Polygon(
62           points={{0,67},{-20,63},{-40,57},{-52,43},{-58,35},{-68,25},{-72,
63               13},{-76,-1},{-78,-15},{-76,-31},{-76,-43},{-76,-53},{-70,-65},
64               {-64,-73},{-48,-77},{-30,-83},{-18,-83},{-2,-85},{8,-89},{22,
65               -89},{32,-87},{42,-81},{54,-75},{56,-73},{66,-61},{68,-53},{
```

```
66              70,-51},{72,-35},{76,-21},{78,-13},{78,3},{74,15},{66,25},{54,
67              33},{44,41},{36,57},{26,65},{0,67}},
68          lineColor={0,0,255},
69          fillColor={80,180,255},
70          fillPattern=FillPattern.Solid),
71        Polygon(
72          points={{-58,35},{-68,25},{-72,13},{-76,-1},{-78,-15},{-76,-31},{
73              -76,-43},{-76,-53},{-70,-65},{-64,-73},{-48,-77},{-30,-83},{-18,

74              -83},{-2,-85},{8,-89},{22,-89},{32,-87},{42,-81},{54,-75},{42,
75              -77},{40,-77},{30,-79},{20,-81},{18,-81},{10,-81},{2,-77},{-12,
76              -73},{-22,-73},{-30,-71},{-40,-65},{-50,-55},{-56,-43},{-58,-35},

77              {-58,-25},{-60,-13},{-60,-5},{-60,7},{-58,17},{-56,19},{-52,
78              27},{-48,35},{-44,45},{-40,57},{-58,35}},
79          lineColor={0,0,255},
80          fillColor={0,128,255},
81          fillPattern=FillPattern.Solid),
82        Ellipse(
83          extent={{-6,-1},{6,-12}},
84          lineColor={255,0,0},
85          fillColor={191,0,0},
86          fillPattern=FillPattern.Solid),
87        Text(
88          extent={{11,13},{50,-25}},
89          lineColor={0,0,0},
90          textString="T"),
91        Line(points={{0,-12},{0,-96}}, color={255,0,0})}),
92    Documentation(info="<HTML>
93 </HTML>
94 "), uses(Modelica(version="3.2")));
95 end AirHeatCapacitor;
```

Listing A.2: Modelica-Code für den AirHeatCapacitor-Block

```
1  model ThermalAirConductor
2    "Lumped thermal element transporting heat without storing it"
3    extends Modelica.Thermal.HeatTransfer.Interfaces.Element1D;
4    parameter Real way( unit="m") "Area of box divided by length of box";
5                                              //results in thermal conductance
                                                  if multiplied by lambda
6    Modelica.SIunits.Temperature T;
7  protected
8    constant Real c2 = -3.795312530606681*10^(-5); //ci..coeff of regpoly of
         lambda
9    constant Real c1 = 0.097175184785803;
10   constant Real c0 = 0.644622988011737;
11 equation
12   T = port_a.T;
13   Q_flow = way*(c2*T^2+c1*T+c0)*10^(-3)*dT;
14   annotation (
15     Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{
16             100,100}}), graphics={
17         Rectangle(
```

```
          extent={{-90,70},{90,-70}},
          lineColor={0,0,255},
          pattern=LinePattern.None,
          fillColor={0,128,255},
          fillPattern=FillPattern.Backward),
        Line(
          points={{-90,70},{-90,-70}},
          color={0,0,0},
          thickness=0.5),
        Line(
          points={{90,70},{90,-70}},
          color={0,0,0},
          thickness=0.5),
        Text(
          extent={{-150,115},{150,75}},
          textString="%name",
          lineColor={0,0,255}),
        Text(
          extent={{-150,-75},{150,-105}},
          lineColor={0,0,0},
          textString="way=%way")}),
    Diagram(coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},
            {100,100}}), graphics={
        Line(
          points={{-80,0},{80,0}},
          color={255,0,0},
          thickness=0.5,
          arrow={Arrow.None,Arrow.Filled}),
        Text(
          extent={{-26,-10},{27,-39}},
          lineColor={255,0,0},
          textString="Q_flow"),
        Text(
          extent={{-80,50},{80,20}},
          lineColor={0,0,0},
          textString="dT = port_a.T - port_b.T")}),
    Documentation(info="
"), uses(Modelica(version="3.2")));
end ThermalAirConductor;
```

Listing A.3: Modelica-Code für den ThermalAirConductor-Block

```
block ThermalAirCompartment
    parameter Real x;
  parameter Real y;
  parameter Real z;
  parameter Real V=x*y*z;
  parameter Real Axy=x*y;
  parameter Real Ayz=y*z;
  parameter Real Axz=x*z;
  parameter Real Tstart = 293.15;

  ThermalAirConductor thermalAirConductor2(way=2*Axz/y)
    annotation (Placement(transformation(
```

```
13        extent={{-10,-10},{10,10}},
14        rotation=-90,
15        origin={-52,-66})));
16    ThermalAirConductor thermalAirConductor3(way=2*Axz/y)
17      annotation (Placement(transformation(
18        extent={{-10,-10},{10,10}},
19        rotation=90,
20        origin={48,70})));
21    ThermalAirConductor thermalAirConductor4(way=2*Axy/z)
22      annotation (Placement(transformation(
23        extent={{-10,-10},{10,10}},
24        rotation=90,
25        origin={-16,66})));
26    ThermalAirConductor thermalAirConductor5(way=2*Axy/z)
27      annotation (Placement(transformation(
28        extent={{-10,-10},{10,10}},
29        rotation=-90,
30        origin={-18,-66})));
31    ThermalAirConductor thermalAirConductor6(way=2*Ayz/x)
32      annotation (Placement(transformation(extent={{52,10},{72,30}})));
33    ThermalAirConductor thermalAirConductor7(way=2*Ayz/x)
34      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
35        rotation=180,
36        origin={-66,-4})));
37    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b1
38      annotation (Placement(transformation(extent={{-10,-110},{10,-90}})));
39    Modelica.Blocks.Interfaces.RealOutput T1
40      annotation (Placement(transformation(extent={{96,-110},{116,-90}})));
41    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b2
42      annotation (Placement(transformation(extent={{-110,-110},{-90,-90}})));
43    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b3
44      annotation (Placement(transformation(extent={{-110,-10},{-90,10}})));
45    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b4
46      annotation (Placement(transformation(extent={{-10,90},{10,110}})));
47    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b5
48      annotation (Placement(transformation(extent={{90,90},{110,110}})));
49    Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b6
50      annotation (Placement(transformation(extent={{90,-10},{110,10}})));
51    Modelica.Thermal.HeatTransfer.Components.Convection convection
52      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
53        rotation=180,
54        origin={-66,20})));
55    Modelica.Thermal.HeatTransfer.Components.Convection convection1 annotation
          (
56        Placement(transformation(
57          extent={{-10,-10},{10,10}},
58          rotation=90,
59          origin={14,66})));
60    Modelica.Thermal.HeatTransfer.Components.Convection convection2 annotation
          (
61        Placement(transformation(
62          extent={{-10,-10},{10,10}},
63          rotation=90,
```

```
64        origin={74,70})));
65    Modelica.Thermal.HeatTransfer.Components.Convection convection3
66      annotation (Placement(transformation(extent={{52,-16},{72,4}})));
67    Modelica.Thermal.HeatTransfer.Components.Convection convection4 annotation
          (
68        Placement(transformation(
69          extent={{-10,-10},{10,10}},
70          rotation=270,
71          origin={6,-66})));
72    Modelica.Thermal.HeatTransfer.Components.Convection convection5 annotation
          (
73        Placement(transformation(
74          extent={{-10,-10},{10,10}},
75          rotation=270,
76          origin={-80,-66})));
77    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor
78      annotation (Placement(transformation(extent={{24,-50},{44,-30}})));
79    AirHeatCapacitor airHeatCapacitor(V=V,Tstart=Tstart)
80      annotation (Placement(transformation(extent={{-14,20},{6,40}})));
81    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor1
82      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
83          rotation=180,
84          origin={-110,-24})));
85    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor2
86      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
87          rotation=270,
88          origin={-50,128})));
89    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor3
90      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
91          rotation=180,
92          origin={138,76})));
93    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor4
94      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
95          rotation=180,
96          origin={130,-22})));
97    Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor5
98      annotation (Placement(transformation(extent={{-10,-10},{10,10}},
99          rotation=0,
100         origin={28,-116})));
101   Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor6
102     annotation (Placement(transformation(extent={{-10,-10},{10,10}},
103         rotation=0,
104         origin={-144,-72})));
105   ConvCoeff convCoeff(
106     position=ConvCoeff.bla.gna.side,
107     l=x*y/(2*(x + y)),
108     A=Axz) annotation (Placement(transformation(extent
          ={{-118,-76},{-98,-56}})));
109   ConvCoeff convCoeff1(
110     position=ConvCoeff.bla.gna.side,
111     l=x*y/(2*(x + y)),
112     A=Ayz) annotation (Placement(transformation(extent={{-134,-20},{-114,0}})
          ));
```

```
113   ConvCoeff convCoeff2(
114     l=z,
115     A=Axy,
116     position=ConvCoeff.bla.gna.up)
117             annotation (Placement(transformation(
118         extent={{-10,-10},{10,10}},
119         rotation=270,
120         origin={-44,94})));
121   ConvCoeff convCoeff3(
122     position=ConvCoeff.bla.gna.side,
123     l=x*y/(2*(x + y)),
124     A=Axz) annotation (Placement(transformation(
125         extent={{-10,-10},{10,10}},
126         rotation=180,
127         origin={104,70})));
128   ConvCoeff convCoeff4(
129     position=ConvCoeff.bla.gna.side,
130     l=x*y/(2*(x + y)),
131     A=Ayz) annotation (Placement(transformation(
132         extent={{-10,-10},{10,10}},
133         rotation=180,
134         origin={96,-28})));
135   ConvCoeff convCoeff5(
136     l=z,
137     A=Axy,
138     position=ConvCoeff.bla.gna.down)
139             annotation (Placement(transformation(
140         extent={{-10,-10},{10,10}},
141         rotation=90,
142         origin={36,-80})));
143 equation
144   connect(thermalAirConductor5.port_b, port_b1)
145                                                 annotation (Line(
146       points={{-18,-76},{-18,-100},{0,-100}},
147       color={191,0,0},
148       smooth=Smooth.None));
149   connect(thermalAirConductor2.port_b, port_b2)
150                                                 annotation (Line(
151       points={{-52,-76},{-52,-100},{-100,-100}},
152       color={191,0,0},
153       smooth=Smooth.None));
154   connect(thermalAirConductor4.port_b, port_b4)
155                                                 annotation (Line(
156       points={{-16,76},{-16,100},{0,100}},
157       color={191,0,0},
158       smooth=Smooth.None));
159   connect(thermalAirConductor3.port_b, port_b5)
160                                                 annotation (Line(
161       points={{48,80},{48,100},{100,100}},
162       color={191,0,0},
163       smooth=Smooth.None));
164   connect(thermalAirConductor6.port_b, port_b6)
165                                                 annotation (Line(
```

```
166       points={{72,20},{100,20},{100,0}},
167       color={191,0,0},
168       smooth=Smooth.None));
169   connect(thermalAirConductor5.port_a, airHeatCapacitor.port)
170                                                             annotation (Line(
171       points={{-18,-56},{-18,21},{-4.2,21}},
172       color={191,0,0},
173       smooth=Smooth.None));
174   connect(thermalAirConductor4.port_a, airHeatCapacitor.port)
175                                                             annotation (Line(
176       points={{-16,56},{-16,21},{-4.2,21}},
177       color={191,0,0},
178       smooth=Smooth.None));
179   connect(thermalAirConductor6.port_a, airHeatCapacitor.port)
180                                                             annotation (Line(
181       points={{52,20},{26,20},{26,21},{-4.2,21}},
182       color={191,0,0},
183       smooth=Smooth.None));
184   connect(thermalAirConductor2.port_a, airHeatCapacitor.port)
185                                                             annotation (Line(
186       points={{-52,-56},{-52,-26},{-4.2,-26},{-4.2,21}},
187       color={191,0,0},
188       smooth=Smooth.None));
189   connect(convection1.fluid, port_b4) annotation (Line(
190       points={{14,76},{14,100},{0,100}},
191       color={191,0,0},
192       smooth=Smooth.None));
193   connect(convection2.fluid, port_b5) annotation (Line(
194       points={{74,80},{74,100},{100,100}},
195       color={191,0,0},
196       smooth=Smooth.None));
197   connect(convection3.fluid, port_b6) annotation (Line(
198       points={{72,-6},{100,-6},{100,0}},
199       color={191,0,0},
200       smooth=Smooth.None));
201   connect(convection3.solid, airHeatCapacitor.port) annotation (Line(
202       points={{52,-6},{26,-6},{26,21},{-4.2,21}},
203       color={191,0,0},
204       smooth=Smooth.None));
205   connect(convection4.solid, airHeatCapacitor.port) annotation (Line(
206       points={{6,-56},{6,21},{-4.2,21}},
207       color={191,0,0},
208       smooth=Smooth.None));
209   connect(convection4.fluid, port_b1) annotation (Line(
210       points={{6,-76},{6,-100},{0,-100}},
211       color={191,0,0},
212       smooth=Smooth.None));
213   connect(convection5.fluid, port_b2) annotation (Line(
214       points={{-80,-76},{-80,-100},{-100,-100}},
215       color={191,0,0},
216       smooth=Smooth.None));
217   connect(convection5.solid, airHeatCapacitor.port) annotation (Line(
218       points={{-80,-56},{-78,-56},{-78,-18},{-16,-18},{-16,21},{-4.2,21}},
```

```
219       color={191,0,0},
220       smooth=Smooth.None));
221   connect(thermalAirConductor7.port_b, port_b3)
222                                                 annotation (Line(
223       points={{-76,-4},{-100,-4},{-100,0}},
224       color={191,0,0},
225       smooth=Smooth.None));
226   connect(thermalAirConductor7.port_a, airHeatCapacitor.port)
227                                                           annotation (Line(
228       points={{-56,-4},{-28,-4},{-28,21},{-4.2,21}},
229       color={191,0,0},
230       smooth=Smooth.None));
231   connect(convection.solid, airHeatCapacitor.port) annotation (Line(
232       points={{-56,20},{-28,20},{-28,21},{-4.2,21}},
233       color={191,0,0},
234       smooth=Smooth.None));
235   connect(convection1.solid, airHeatCapacitor.port) annotation (Line(
236       points={{14,56},{14,44},{8,44},{8,21},{-4.2,21}},
237       color={191,0,0},
238       smooth=Smooth.None));
239   connect(temperatureSensor.port, airHeatCapacitor.port) annotation (Line(
240       points={{24,-40},{12,-40},{12,21},{-4.2,21}},
241       color={191,0,0},
242       smooth=Smooth.None));
243   connect(temperatureSensor.T, T1) annotation (Line(
244       points={{44,-40},{54,-40},{54,-100},{106,-100}},
245       color={0,0,127},
246       smooth=Smooth.None));
247
248   connect(convection2.solid, airHeatCapacitor.port) annotation (Line(
249       points={{74,60},{74,44},{42,44},{42,21},{-4.2,21}},
250       color={191,0,0},
251       smooth=Smooth.None));
252   connect(thermalAirConductor3.port_a, airHeatCapacitor.port)
253                                                           annotation (Line(
254       points={{48,60},{48,50},{32,50},{32,21},{-4.2,21}},
255       color={191,0,0},
256       smooth=Smooth.None));
257   connect(convection.fluid, port_b3) annotation (Line(
258       points={{-76,20},{-100,20},{-100,0}},
259       color={191,0,0},
260       smooth=Smooth.None));
261   connect(temperatureSensor5.port, port_b1) annotation (Line(
262       points={{18,-116},{12,-116},{12,-100},{0,-100}},
263       color={191,0,0},
264       smooth=Smooth.None));
265   connect(temperatureSensor1.port, port_b3) annotation (Line(
266       points={{-100,-24},{-100,0}},
267       color={191,0,0},
268       smooth=Smooth.None));
269   connect(temperatureSensor6.T, convCoeff.u2) annotation (Line(
270       points={{-134,-72},{-120,-72}},
271       color={0,0,127},
```

```
272       smooth=Smooth.None));
273   connect(convCoeff1.u2, temperatureSensor1.T) annotation (Line(
274       points={{-136,-16},{-140,-16},{-140,-24},{-120,-24}},
275       color={0,0,127},
276       smooth=Smooth.None));
277   connect(convCoeff.y, convection5.Gc) annotation (Line(
278       points={{-97,-66},{-70,-66}},
279       color={0,0,127},
280       smooth=Smooth.None));
281   connect(convCoeff1.y, convection.Gc) annotation (Line(
282       points={{-113,-10},{-66,-10},{-66,10}},
283       color={0,0,127},
284       smooth=Smooth.None));
285   connect(temperatureSensor2.T, convCoeff2.u2) annotation (Line(
286       points={{-50,118},{-50,106}},
287       color={0,0,127},
288       smooth=Smooth.None));
289   connect(convCoeff2.y, convection1.Gc) annotation (Line(
290       points={{-44,83},{-44,66},{4,66}},
291       color={0,0,127},
292       smooth=Smooth.None));
293   connect(temperatureSensor3.T, convCoeff3.u2) annotation (Line(
294       points={{128,76},{116,76}},
295       color={0,0,127},
296       smooth=Smooth.None));
297   connect(convCoeff3.y, convection2.Gc) annotation (Line(
298       points={{93,70},{64,70}},
299       color={0,0,127},
300       smooth=Smooth.None));
301   connect(temperatureSensor4.T, convCoeff4.u2) annotation (Line(
302       points={{120,-22},{108,-22}},
303       color={0,0,127},
304       smooth=Smooth.None));
305   connect(convCoeff4.y, convection3.Gc) annotation (Line(
306       points={{85,-28},{74,-28},{74,4},{62,4}},
307       color={0,0,127},
308       smooth=Smooth.None));
309   connect(convCoeff5.u2, temperatureSensor5.T) annotation (Line(
310       points={{42,-92},{42,-116},{38,-116}},
311       color={0,0,127},
312       smooth=Smooth.None));
313   connect(convCoeff5.y, convection4.Gc) annotation (Line(
314       points={{36,-69},{36,-66},{16,-66}},
315       color={0,0,127},
316       smooth=Smooth.None));
317   connect(convCoeff5.u1, temperatureSensor.T) annotation (Line(
318       points={{30,-92},{30,-100},{54,-100},{54,-40},{44,-40}},
319       color={0,0,127},
320       smooth=Smooth.None));
321   connect(convCoeff.u1, temperatureSensor.T) annotation (Line(
322       points={{-120,-60},{-132,-60},{-132,-132},{50,-132},{50,-40},{44,-40}},
323       color={0,0,127},
324       smooth=Smooth.None));
```

```
325   connect(convCoeff1.u1, temperatureSensor.T) annotation (Line(
326       points={{-136,-4},{-146,-4},{-146,-40},{44,-40}},
327       color={0,0,127},
328       smooth=Smooth.None));
329   connect(convCoeff4.u1, temperatureSensor.T) annotation (Line(
330       points={{108,-34},{116,-34},{116,-48},{66,-48},{66,-40},{44,-40}},
331       color={0,0,127},
332       smooth=Smooth.None));
333   connect(convCoeff3.u1, temperatureSensor.T) annotation (Line(
334       points={{116,64},{152,64},{152,-64},{60,-64},{60,-40},{44,-40}},
335       color={0,0,127},
336       smooth=Smooth.None));
337
338   connect(convCoeff2.u1, temperatureSensor.T) annotation (Line(
339       points={{-38,106},{-38,116},{20,116},{20,-18},{44,-18},{44,-40}},
340       color={0,0,127},
341       smooth=Smooth.None));
342   connect(port_b3, port_b3) annotation (Line(
343       points={{-100,0},{-100,0}},
344       color={191,0,0},
345       smooth=Smooth.None));
346   connect(temperatureSensor2.port, port_b4) annotation (Line(
347       points={{-50,138},{0,138},{0,100}},
348       color={191,0,0},
349       smooth=Smooth.None));
350   connect(temperatureSensor4.port, port_b6) annotation (Line(
351       points={{140,-22},{148,-22},{148,0},{100,0}},
352       color={191,0,0},
353       smooth=Smooth.None));
354   connect(temperatureSensor6.port, port_b2) annotation (Line(
355       points={{-154,-72},{-158,-72},{-158,-100},{-100,-100}},
356       color={191,0,0},
357       smooth=Smooth.None));
358   connect(temperatureSensor3.port, port_b5) annotation (Line(
359       points={{148,76},{152,76},{152,100},{100,100}},
360       color={191,0,0},
361       smooth=Smooth.None));
362   annotation (uses(Modelica(version="3.2")), Diagram(graphics),
363     Icon(graphics={Rectangle(
364           extent={{-100,100},{100,-100}},
365           lineColor={0,128,255},
366           fillPattern=FillPattern.Sphere,
367           fillColor={255,255,255}),
368     Text( extent={{-71,20},{71,-20}},
369           textString="%name",
370           lineColor={0,0,255})}));
371 end ThermalAirCompartment;
```

Listing A.4: Modelica-Code für den ThermalAirCompartment-Block

```
1 block ThermalSolidCompartment
2   Modelica.Thermal.HeatTransfer.Components.HeatCapacitor heatCapacitor(C=cp*m
        );
3   parameter Real x;
```

```
4   parameter Real y;
5   parameter Real z;
6   parameter Real V=x*y*z;
7   parameter Real Axy=x*y;
8   parameter Real Ayz=y*z;
9   parameter Real Axz=x*z;
10  parameter Real cp=1000;
11  parameter Real rho=200;
12  parameter Real m=V*rho;
13  parameter Real k=2.1;
14
15  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor2
        (G=2*k*Axz
16        /y)
17    annotation (Placement(transformation(
18        extent={{-10,-10},{10,10}},
19        rotation=-90,
20        origin={-52,-66})));
21  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor3
        (G=2*k*Axz
22        /y)
23    annotation (Placement(transformation(
24        extent={{-10,-10},{10,10}},
25        rotation=90,
26        origin={48,70})));
27  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor4
        (G=2*k*Axy
28        /z)
29    annotation (Placement(transformation(
30        extent={{-10,-10},{10,10}},
31        rotation=90,
32        origin={-14,54})));
33  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor5
        (G=2*k*Axy
34        /z)
35    annotation (Placement(transformation(
36        extent={{-10,-10},{10,10}},
37        rotation=-90,
38        origin={0,-62})));
39  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor6
        (G=2*k*Ayz
40        /x)
41    annotation (Placement(transformation(extent={{54,-10},{74,10}})));
42  Modelica.Thermal.HeatTransfer.Components.HeatCapacitor heatCapacitor1(C=cp*
        m, T(start=
43          293.15))
44    annotation (Placement(transformation(extent={{-10,0},{10,20}})));
45  Modelica.Thermal.HeatTransfer.Components.ThermalConductor thermalConductor7
        (G=2*k*Ayz
46        /x)
47    annotation (Placement(transformation(extent={{-10,-10},{10,10}},
48        rotation=180,
49        origin={-70,0})));
```

```
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b1
    annotation (Placement(transformation(extent={{-10,-110},{10,-90}})));
  Modelica.Blocks.Interfaces.RealOutput T1
    annotation (Placement(transformation(extent={{100,-110},{120,-90}})));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b2
    annotation (Placement(transformation(extent={{-110,-110},{-90,-90}})));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b3
    annotation (Placement(transformation(extent={{-110,-10},{-90,10}})));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b4
    annotation (Placement(transformation(extent={{-10,90},{10,110}})));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b5
    annotation (Placement(transformation(extent={{90,90},{110,110}})));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b port_b6
    annotation (Placement(transformation(extent={{90,-10},{110,10}})));
  Modelica.Thermal.HeatTransfer.Sensors.TemperatureSensor temperatureSensor
    annotation (Placement(transformation(extent={{24,-50},{44,-30}})));
equation
  connect(thermalConductor5.port_b,port_b1)  annotation (Line(
      points={{-1.83697e-015,-72},{-1.83697e-015,-100},{0,-100}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor2.port_b,port_b2)  annotation (Line(
      points={{-52,-76},{-52,-100},{-100,-100}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor4.port_b,port_b4)  annotation (Line(
      points={{-14,64},{-14,68},{0,68},{0,72},{0,72},{0,100},{0,100}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor3.port_b,port_b5)  annotation (Line(
      points={{48,80},{48,100},{100,100}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor6.port_b,port_b6)  annotation (Line(
      points={{74,0},{100,0}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor5.port_a,heatCapacitor1. port) annotation (Line(
      points={{1.83697e-015,-52},{1.83697e-015,-40},{0,-40},{0,0}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor4.port_a,heatCapacitor1. port) annotation (Line(
      points={{-14,44},{-14,0},{0,0}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor6.port_a,heatCapacitor1. port) annotation (Line(
      points={{54,0},{0,0}},
      color={191,0,0},
      smooth=Smooth.None));
  connect(thermalConductor2.port_a, heatCapacitor1.port) annotation (Line(
      points={{-52,-56},{-52,-26},{0,-26},{0,0}},
      color={191,0,0},
      smooth=Smooth.None));
```

```
103   connect(thermalConductor7.port_b, port_b3) annotation (Line(
104       points={{-80,1.22465e-015},{-88,1.22465e-015},{-88,0},{-100,0}},
105       color={191,0,0},
106       smooth=Smooth.None));
107   connect(thermalConductor7.port_a, heatCapacitor1.port) annotation (Line(
108       points={{-60,-1.22465e-015},{-29,-1.22465e-015},{-29,0},{0,0}},
109       color={191,0,0},
110       smooth=Smooth.None));
111   connect(temperatureSensor.port, heatCapacitor1.port) annotation (Line(
112       points={{24,-40},{12,-40},{12,0},{0,0}},
113       color={191,0,0},
114       smooth=Smooth.None));
115   connect(temperatureSensor.T, T1) annotation (Line(
116       points={{44,-40},{54,-40},{54,-100},{110,-100}},
117       color={0,0,127},
118       smooth=Smooth.None));
119
120   connect(thermalConductor3.port_a, heatCapacitor1.port) annotation (Line(
121       points={{48,60},{48,50},{32,50},{32,0},{0,0}},
122       color={191,0,0},
123       smooth=Smooth.None));
124   connect(T1, T1) annotation (Line(
125       points={{110,-100},{110,-100}},
126       color={0,0,127},
127       smooth=Smooth.None));
128   annotation (uses(Modelica(version="3.2")), Diagram(graphics),
129     Icon(graphics={Rectangle(
130           extent={{-100,100},{100,-100}},
131           lineColor={54,17,0},
132           fillColor={207,103,0},
133           fillPattern=FillPattern.Sphere),
134     Text( extent={{-71,20},{71,-20}},
135           textString="%name",
136           lineColor={54,17,0})}));
137 end ThermalSolidCompartment;
```

Listing A.5: Modelica-Code für den ThermalSolidCompartment-Block, weitgehend aus [Rö12] entnommen

Abbildungsverzeichnis

Literaturverzeichnis

[Assa] Modelica Association. Functional mockup interface. https://www.fmi-standard.org/. Zugriff: 25.11.2012.

[Assb] Modelica Association. Modelica. https://www.modelica.org/. Zugriff: November 2012.

[CG91] F.E. Cellier and J. Greifeneder. *Continuous System Modeling*. Springer, 1991.

[CK06] F.E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 2006.

[Con] Open Source Modelica Consortium. Openmodelica. https://www.openmodelica.org/. Zugriff: November 2012.

[Fri04] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons, 2004.

[GC06] V. Gesellschaft and VDI-Gesellschaft Verfahrenstechnik Und Chemieingenieurwesen. *VDI-Wärmeatlas*. Number Bd. 1 in Vdi-buch. Springer, 2006.

[Hei12] Bernhard Heinzl. Objektorientierte multi-domain-modellierung und simulation von werkzeugmaschinen, 2012.

[ITI] ITI. Simulationx. http://www.itisim.com/simulationx/. Zugriff: November 2012.

[KS00] R. Kübler and W. Schiehlen. Two methods of simulator coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):93–113, 2000.

[Lab] Berkeley National Laboratory. Energyplus documentation. http://apps1.eere.energy.gov/buildings/energyplus/energyplus_documentation.cfm. Zugriff: Dezember 2012.

[Map] Maplesoft. Maplesim. http://www.maplesoft.com/products/maplesim/index.aspx. Zugriff: 18.11.2012.

[Mat] MathWorks. Mathworks produkte und dienstleistungen. http://www.mathworks.de/products/. Zugriff: November 2012.

[Mel08] Markus Melenk. Numerik von differentialgleichungen, 2008. Vorlesungsskriptum zur LVA Numerik von Differentialgleichungen.

[otUoC] The Regents of the University of California. Ptolemy ii. `http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm`. Zugriff: 18.11.2012.

[Rö12] Matthias Rößler. Modellkopplung thermodynamischer systeme, 2012.

[SFR04] W.R. Stevens, B. Fenner, and A.M. Rudoff. *Unix Network Programming: The Sockets Networking Api*. Number Bd. 1 in Addison-Wesley Professional Computing Series. Addison-Wesley, 2004.

[Sys] Dassault Systems. Dymola. `http://www.3ds.com/products/catia/portfolio/dymola/overview/`. Zugriff: November 2012.

[Tr8] Marija Trčka. Co-simulation for performance prediction of innovative integrated mechanical energy systems in buildings, 2008.

[TS12] Christoph Clauss Tom Schierz, Martin Arnold. Co-simulation with communication step size control in an fmi compatible master algorithm. In *Proceedings of the 9th International MODELICA Conference*, September 3-5 2012.

[Völ10] L. Völker. *Untersuchung des Kommunikationsintervalls bei der gekoppelten Simulation*. Karlsruher Schriftenreihe Fahrzeugsystemtechnik. KIT Scientific Publishing, 2010.

[Wet11] Michael Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. `http://simulationresearch.lbl.gov/wetter/download/LBNL-81118.pdf`, August 31 2011. Zugriff: 21.11.2012.

[Wet12] Michael Wetter. Building controls virtual test bed user manual. `http://simulationresearch.lbl.gov/bcvtb/releases/1.2.0/doc/manual/bcvtb-manual.pdf`, October 23 2012. Zugriff: 18.11.2012.

[Wol] Wolfram. Wolfram systemmodeler. `http://www.wolfram.com/system-modeler/`. Zugriff: November 2012.