TECHNISCHE UNIVERSITÄT
TUHH
HAMBURG — HARBURG

Arbeitsbereich Energietechnik

Prof. Dr.-Ing. G. Schmitz

DIPLOMARBEIT

Erstellung von Grundmodellen zur Simulation von Dampferzeugern in der objektorientierten Simulationsumgebung SMILE

Autor: cand. ing. Oliver Czygan

Betreuer: Dipl.-Ing. Georg Mühlthaler

Ich versichere, diese Arbeit im Rahmen der am Arbeitsbereich üblichen Betreuung selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Diese Arbeit ist nach bestem Wissen erstellt worden; für den Inhalt kann jedoch keine Gewähr übernommen werden.

Hamburg, im März 1997

Arbeitsbereich Energietechnik

TECHNISCHE UNIVERSITÄT HAMBURG — HARBURG

Herrn cand. ing. Oliver Czygan
Matrikelnummer: 2492

Aufgabenstellung für eine Diplomarbeit

Thema:

Erstellung von Grundmodellen zur Simulation von Dampferzeugern in der objektorientierten Simulationsumgebung Smile

Im Rahmen eines Projektes zur dynamischen Simulation von GuD-Kraftwerken sind in dieser Arbeit Modelle zur dynamischen Beschreibung von Dampferzeugern zu erstellen.

Als grundlegende Elemente jedes Dampferzeugers können fluiddurchströmte Kontrollvolumen, Wärmeübergänge und das Verhalten von Wasserdampf angesehen werden. Ein Schwerpunkt der Arbeit liegt deshalb auf der Modellierung dieser Elemente.

- Gleichungen für Energie, Masse und Impuls für Kontrollvolumen:
  Beschreibung der Grundgleichungen, Annahmen und Vereinfachungen, der Diskretisierung und der endgültig implementierten Form.
- Erstellung einer Klassenbibliothek für Wärmeübertragung:
  Konvektion, Verdampfung, instationäre Wärmeleitung in (Rohr-)Wänden, Strahlungswärmeübergang.
- Untersuchung, wie Routinen zur Berechnung der Zustandsgrößen von Wasser/Wasserdampf geeignet angebunden werden können.

Desweiteren sollen mögliche Modellierungen für Eco, Verdampfer und Überhitzer programmiert und getestet werden. Besonderheiten und Schwierigkeiten bei der Implementation sollen aufgezeigt und diskutiert werden. Kriterien hierfür sind: Sinnvolle Klassenstruktur sowie numerische Aspekte (Stabilität der Rechnung, Einfluß der Anfangswerte, Rechenzeit und Genauigkeit, Steifigkeit des DGL–Systems). Die Anbindung eines Einspritzkühlers sowie die Umlaufart sollen ebenfalls Berücksichtigung finden.

Die Modelle sind in das Simulationssystem Smile zu implementieren und so zu gestalten, daß sie Bestandteil einer ausbaufähigen objektorientierten Modellbibliothek werden. Dem objektorientierten Charakter von Smile ist dabei besonders Rechnung zu tragen.

Die Modelle sollen verständlich dokumentiert werden, leicht in die Beschreibung einer gesamten Kraftwerksanlage einzufügen sein und geeignete Schnittstellen zur Einbindung in ein Regelungskonzept aufweisen.

Hamburg, den

Prof. Dr.-Ing. G. Schmitz Dipl.-Ing. G. Mühlthaler

Inhaltsverzeichnis

Anhang

Formelzeichen und Indizes

Formelzeichen

Symbol	in Programmen	Einheit	Bedeutung
a	`a`	m^2/s	Temperaturleitfähigkeit
A	`A`	m^2	Fläche
c	`c`	m/s	Geschwindigkeit
c_p	`cp`	J/kgK	spezifische Wärmekapazität bei konst. Druck
C_S	`C_S`	W/m^2K^4	Strahlungskonstante; $C_S = 5,67 \cdot 10^{-8}$ W/m^2K^4
d	`d`	m	Durchmesser
f	`f`	—	Flächenfaktor
h	`h`	J/kg	massenspezifische Enthalpie
H_u	`H_u`	kJ/kg	Heizwert
k	`k`	W/m^2K	Wärmedurchgangskoeffizient
Ko	`Ko`	—	Konakowzahl
l	`l`	m	Länge
L	—	m	Länge
m	`mass`	kg	Masse
$\dot{m}$	`m_dot`	kg/s	Massenstrom
n	`n`	—	Anzahl
ND	`ND`	—	Anzahl der Diskretisierungselemente (in Richtung Rohrachse)
ND_WL	`ND_WL`	—	Anzahl der Diskretisierungselemente (senkrecht zur Rohrachse)
Nu	`Nu`	—	Nusseltzahl
p	`p`	bar, Pa	Druck
Pe	—	—	Pecletzahl
Pr	`Pr`	—	Prandtlzahl
$\dot{q}$	`q_dot`	W/m^2	Wärmestromdichte
$\dot{Q}$	`Q_dot`	W	Wärmeleistung
r	`r`	J/kg	massenspezifische Verdampfungsenthalpie
r	`r`	m	Koordinate in rohrachssenkrechter Richtung
R	`R`	J/kgK	Gaskonstante
Re	`Re`	—	Reynoldszahl
s	`s`	m	Wandstärke
S	—	—	Schlupf
t	`time`	s	Zeit
T	`T`	K	Temperatur
v	`v`	m^3/kg	massenspezifisches Volumen
V	`V`	m^3	Volumen
W	—	W/kg	massenspezifische Wärmeleistung
x	`x`	m	Koordinate in Rohrachsenrichtung
x	`x`	—	Dampfgehalt
Δx	`delta_x`	m	Länge eines Diskretisierungselements
$\dot{x}$	`x_dot`	—	Dampfmassenstromanteil
z	`z`	—	konventioneller Schwärzefaktor
z	—	m	Höhe
α	`alpha`	W/m^2K	Wärmeübergangskoeffizient
β	`beta`	rad	Steigungswinkel Verdampferrohr
Γ	—	verschied.	Diffusionskoeffizient

Symbol	in Programmen	Einheit	Bedeutung
ε	`epsilon`	—	Emissionskoeffizient
η	—	10^{-6} kg/ms	dynamische Viskosität
ϑ	`theta`	°C	Temperatur
Θ	—	—	dimensionsloses Temperaturverhältnis
λ	`lambda`	W/mK	Wärmeleitfähigkeit
λ_R	`lambda_r`	—	Druckverlustbeiwert
ν	—	m^2/s	kinematische Viskosität
ξ	`xi`	—	Druckverlustbeiwert
Π	—	—	Druckverhältnis
ϱ	`rho`	kg/m^3	Dichte
τ	`TAU`	s	Zeitkonstante
υ	—	%	mittlere relative Streuung
Φ	—	verschied.	abhängige Variable der allgemeinen Transportgleichung
ψ	`psi`	—	Hohlraumanteil
ω	`omega`	1/s	Kreisfrequenz

Abkürzungen und Indizes

Symbol	in Programmen	Bedeutung
1	—	Einzelrohr
1Ph	—	Einphasengebiet
2Ph	—	Zweiphasengebiet
a	`a`	außen
A	—	Asche
A	—	Anordnung
ad	`_ad`	adiabat
Br	`_br`	Brennstoff
CM	`CM`	konzentrierte Masse (Concentrated Mass)
DM	`DM`	verteilte Masse (Distributed Mass)
E	—	östlich
FA	`_FA`	Feuerraumaustritt
g	`_g`	Rauchgas
h	—	hydraulisch
hilf	`_hilf`	Hilfs ...
i	`i`	innen
i	`i`	Laufindex bei Diskretisierung in Rohrachsrichtung
in	`_in`	Eintritt
j	`j`	Laufindex bei Rohrwanddiskretisierung in radialer Richtung
konv	`_konv`	konvektiv
krit	`KRIT`	kritisch
l	`l`	flüssig
lam	`_lam`	laminar
L	`_l`	Luft
m	`m`	Mittelwert
out	`_out`	Austritt
q	`q`	Querschnitts ...
R	`r`	Reibung
Rb	—	Rohrbündel
Rg	`_rg`	Rauchgas
S	`_s`	Siede ...
stat	`_stat`	stationär
strlg	`_strlg`	Strahlung
SWF	`SWF`	Stoffwertefunktion
tubes	`tubes`	Rohre
turb	`_turb`	turbulent
Üg	—	Übergangsgebiet
unv	—	unverbrannt
vI	—	vor Integrator
w	`_w`	Wasser
W	`_wall`	Wand
W	—	westlich
WL	`_WL`	Wärmeleitung
$'$	`_1`	Siedezustand
$''$	`_2`	Sattdampfzustand
$*$	—	bezogene Größe
$\dot{}$	`_dot`	Strömungsgröße

Mathematische Symbole

Symbol	**Bedeutung**
∂x	Differential der Variable x
Δx	Differenz der Variable x
$f(x)$	Funktion von x
$\|x\|$	Betrag von x
$[[x;y]]$	größter Wert von x und y
$\overline{x}$	Mittelwert von x
$\widehat{x}$	Amplitude von x
∇u	Nabla-Operator; $\nabla u = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}$

Notationskonventionen

Um die Ausarbeitungen möglichst übersichtlich zu gestalten, sind hervorzuhebende Textpassagen gemäß folgenden Konventionen dargestellt:

- Klassennamen werden in serifenloser Schrift gesetzt: WaermeleitungRohr
- Programmlistings werden durch den Schrifttyp Courier gekennzeichnet, SMILE-Schlüsselwörter innerhalb des Programmcodes sind fett hervorgehoben:
  `@eq cont Q_dot_rg (m_dot_rg, cp, t_rg_in, t_rg_out)`
- Dateinamen werden herausgestellt durch Verwendung von Schreibmaschinenschrift in Anführungszeichen: „`RgWasserWAT.smi`“
- Bezeichnungen für Stoffwertfunktionen des Programmpakets PROST sind durch Verwendung von Schreibmaschinenschrift hervorgehoben, z. B. `sat_t`
- Programmnamen werden in Kapitälchen geschrieben: SMILE
- Sonstige Hervorhebungen werden durch Verwendung kursiver Schrift dargestellt: *normale Hervorhebung*

1 Einleitung

Die Forderung nach Anlagensicherheit und Wirtschaftlichkeit prägt die gesamte Dampferzeugertechnik. Gerade eine weitere Wirkungsgradsteigerung ist aufgrund eines gestiegenen Umwelt- und Energiebewußtseins das Ziel zukünftiger Entwicklung.

Während zur Beurteilung eines Kraftwerksblockes, der im Grundlastbetrieb gefahren wird, stationäre Betriebspunkte sowie An- und Abfahrvorgänge zu betrachten sind, interessieren bei Dampferzeugern in Mittel- und Spitzenlastkraftwerken zusätzlich die instationären Zustände, wie z. B. Laständerungen bzw. Anpassung an schwankende Lastanforderungen. Sie beanspruchen den Dampferzeuger i. a. stärker als im stationären Betriebszustand und wirken lebensdauermindernd. Für Dampferzeuger in Mittel- und Spitzenlastkraftwerken werden besonders starke Anforderungen an die Regelbarkeit und das Teillastverhalten gestellt, sie erfordern insbesondere eine materialschonende und wirtschaftliche Fahrweise. Zur besseren Ausnutzung der z. B. von den verwendeten Werkstoffen gesetzten zulässigen Grenzdaten ist eine genaue Kenntnis des Verhaltens der einzelnen Komponenten sowie des Zusammenwirkens im gesamten System erforderlich.

Leistungsfähige Simulationsprogramme bieten wichtige Hilfen für die Konzeption und Auslegung energietechnischer Anlagen. Mit der Möglichkeit zur Beurteilung instationärer Betriebszustände liefern dynamische Rechenmodelle und -programme bedeutende Aussagen über die thermische Belastung der Bauteile und die Einhaltung von Forderungen an den dynamischen Betrieb.

Häufig ist nicht nur eine bestehende Anlage mit einem konkreten Aufbau zu simulieren. Sollen vielmehr einzelne wiederverwend- und erweiterbare Komponenten zur beliebigen Kombination in einer Modellbibliothek zur Verfügung gestellt werden, so bietet die objektorientierte Programmierung gegenüber der prozeduralen deutliche Vorteile. Ein Werkzeug zur dynamischen Simulation energiewandelnder Systeme ist die an der TU Berlin entwickelte objektorientierte Simulationsumgebung SMILE. Sie beinhaltet eine auf Objective-C basierende Programmiersprache, eine Laufzeitumgebung sowie eine Reihe von mathematischen Lösern.

Ziel dieser Arbeit ist es, für SMILE grundlegende Elemente von Dampferzeugern zu modellieren. Hierzu zählen fluiddurchströmte Kontrollvolumen und Klassen zur Beschreibung der Wärmeübertragung. Nach einer Betrachtung, wie Stoffwertroutinen geeignet einzubinden sind, werden konkrete Dampferzeugerkomponenten unter Berücksichtigung von Rechenzeitbedarf und -genauigkeit simuliert. Beachtet werden muß schon hier die Möglichkeit der späteren Kombination der Einzelbestandteile zu einem Gesamtsystem. Die Arbeit endet nach einer Betrachtung spezieller numerischer Gesichtspunkte mit einigen Rechenzeitvergleichen.

2 Basisklassen

Bevor Dampferzeugerbauteile modelliert werden, sind hierfür erforderliche Basisklassen zur Beschreibung der Transportgleichungen, der Wärmeübertragungs- und -speicherungsbeziehungen und des Druckabfalls in durchströmten Volumen bereitzustellen. Zunächst wird eine allgemeine Form der Bilanzgleichungen formuliert. Größere Bauteile werden in finite Bestandteile zerlegt, und für jedes Diskretisierungselement sind entsprechende Beziehungen aufzustellen.

2.1 Allgemeine Differentialgleichung

Für Kontrollvolumen sollen die grundlegenden physikalischen Erhaltungsgleichungen formuliert werden.

Die Bilanzgleichungen für Masse, Energie und Impuls können auf einen gemeinsamen Typ zurückgeführt werden. Die allgemeine Transportgleichung lautet:

$$\underbrace{\frac{\partial}{\partial t}(\varrho \cdot \Phi)}_{\text{Speicherung}} + \underbrace{\frac{\partial}{\partial x}(\varrho \cdot c \cdot \Phi)}_{\text{Konvektion}} = \underbrace{\frac{\partial}{\partial x}\left(\Gamma_\Phi \cdot \frac{\partial \Phi}{\partial x}\right)}_{\text{Diffusion}} + \underbrace{S_\Phi}_{\text{Quellterm}} \tag{2.1}$$

Hierin bezeichnen ϱ die Dichte, c die Geschwindigkeit, Γ den Diffusionskoeffizienten, S_Φ den Quellterm — in S_Φ werden alle Terme, die nicht zur Konvektion oder Diffusion gehören, vereint — und $\frac{\partial}{\partial t}$ und $\frac{\partial}{\partial x}$ die Ableitung nach der Zeit bzw. dem Ort. Die abhängige Variable Φ ist eine Funktion des Ortes und der Zeit: $\Phi = \Phi(x, y, z, t)$. Die abhängige Variable Φ und der Diffusionskoeffizient Γ haben für die unterschiedlichen Erhaltungssätze gemäß Tabelle 4 verschiedene Bedeutungen.

Erhaltungsgröße	Variable Φ	Diffusionskoeffizient Γ
Masse	1	0
Energie	h	$\frac{\lambda}{c_p}$
Impuls	c	η

Tabelle 4: Abhängige Variable Φ und Diffusionskoeffizient Γ für die unterschiedlichen Erhaltungsgleichungen

Berücksichtigt man in Gleichung (2.1) den Sonderfall der Massenerhaltung ($\Phi = 1$; $\Gamma = 0$; $S_\Phi = 0$. Siehe Gleichung (2.9) in Kapitel 2.3.1), so ist folgende vereinfachte Schreibweise in der für SMILE gewünschten Form $\frac{\partial}{\partial t}(\text{Variable}) = \ldots$ möglich:

$$\frac{\partial \Phi}{\partial t} = -c \cdot \frac{\partial \Phi}{\partial x} + \frac{1}{\varrho} \cdot \frac{\partial}{\partial x}\left(\Gamma_\Phi \cdot \frac{\partial \Phi}{\partial x}\right) + \frac{1}{\varrho} \cdot S_\Phi \tag{2.2}$$

2.2 Diskretisierung

Zur numerischen Lösung der allgemeinen Transportgleichung (2.1) bzw. (2.2) ist eine Untergliederung in Teilvolumen erforderlich. In der Simulationsumgebung SMILE [1] ist eine Diskretisierung nur bei Orts-, nicht aber bei Zeitableitungen notwendig — unter „Diskretisierung“ ist im folgenden also immer eine *Orts*diskretisierung zu verstehen. Es ist eine Vorschrift

[1]Weiterführende Informationen zur Simulationsumgebung SMILE stehen z. B. in [7] und in [58]

bereitzustellen, wo welche Größen definiert sind und wie die Diskretisierung vorzunehmen ist.

Zur Lösung der allgemeinen Differentialgleichung wird das von Patankar in [45] vorgeschlagene Finite-Volumen-Verfahren verwendet. Die Pecletzahl

$$\mathrm{Pe} = \frac{\varrho \cdot c \cdot \Delta x}{\Gamma} \tag{2.3}$$

beschreibt das Verhältnis zwischen der Stärke der Konvektion und der der Diffusion eines durchströmten Kontrollvolumens. Für Pecletzahlen $\mathrm{Pe} > 10$ liefert nach [45] das „Aufwärtsdifferenzenverfahren“ bessere Ergebnisse als das „Zentraldifferenzenverfahren“. Da in Dampferzeugern i. a. mit großen Pecletzahlen zu rechnen ist, wird die Diskretisierung nach dem Aufwärtsdifferenzenschema vorgenommen. Das Zentraldifferenzenverfahren berücksichtigt bei der Bestimmung der Geschwindigkeit oder des Massenstroms im Element selbst die jeweiligen Größen an beiden Elementrändern, die Diskretisierung ist unabhängig von der Strömungsrichtung. Dieses Schema liefert für Pecletzahlen $\mathrm{Pe} \approx 1$ bessere Ergebnisse als das Aufwärtsdifferenzenverfahren.

Die Zustandsgrößen Enthalpie, Temperatur, Druck und Dichte werden im Kontrollvolumen — sowohl in Strömungsrichtung als auch über den Querschnitt — konstant angenommen. Geschwindigkeit und Massenstrom sind an den Querschnittsflächen zwischen zwei Diskretisierungselementen und am Ein- und Austritt definiert. Nach dem Aufwärtsdifferenzenschema wird als Geschwindigkeit bzw. Massenstrom im diskreten Element selbst die jeweilige Größe am Elementeintritt verwendet. Je nach Strömungsrichtung ist also mit dem westlichen (W) bzw. dem östlichen (E) Wert zu rechnen.

Die Bezeichnung der Elemente bzw. der Elementgrenzen eines durchströmten Rohres ist grafisch in Abbildung 1 dargestellt, das betrachtete Bauteil wird gleichmäßig in Elemente der Länge

$$\Delta x = \frac{l}{\mathrm{ND}} \tag{2.4}$$

unterteilt.

Benutzt man die in Abbildung 2 an einem einzelnen Diskretisierungselement veranschaulichte „Kompaßnotation“, so lautet mit

$$a_E = \frac{\Gamma_e}{(\delta x)_e} + [[-(\varrho \cdot c)_e; 0]], \tag{2.5}$$

$$a_W = \frac{\Gamma_w}{(\delta x)_w} + [[(\varrho \cdot c)_w; 0]] \tag{2.6}$$

und

$$a_P = a_E + a_W \tag{2.7}$$

die diskretisierte Form der allgemeinen Transportgleichung:

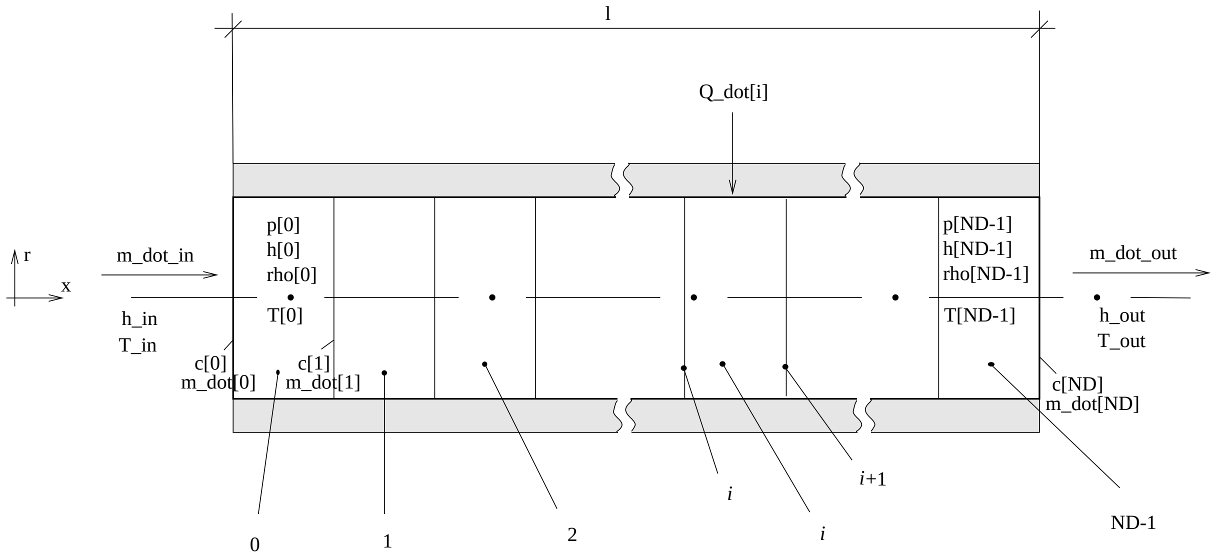

Abbildung 1: Rohrdiskretisierung in Achsrichtung

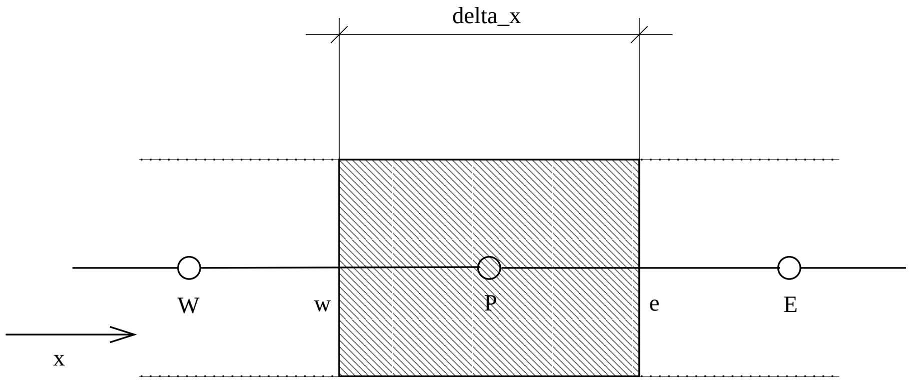

Abbildung 2: Kontrollvolumen für Diskretisierung

$$\frac{\partial \Phi_P}{\partial t} = \frac{1}{\varrho \cdot \Delta x} \cdot (a_E \cdot \Phi_E + a_W \cdot \Phi_W - a_P \cdot \Phi_P + S_\Phi \cdot \Delta x) \tag{2.8}$$

Hierin bezeichnet der Index P das Element selbst, W bzw. E das westlich bzw. östlich angrenzende Element und w bzw. e die entsprechenden Grenzflächen.

Die für Masse-, Energie- und Impulsgleichung erforderliche Zerlegung in finite Kontrollvolumen wird für SMILE in der grundlegenden Bilanzierungsklasse BalanceBaseClass vorgenommen. Je nachdem, ob mit konzentrierter oder verteilter Masse gerechnet werden soll, stellen die von BalanceBaseClass erbenden Klassen BalanceBaseClassCM bzw. BalanceBaseClassDM die zusätzlichen Schnittgrößen `c` und `rho` bzw. `c[i]` und `rho[i]` bereit. Alle Klassen der Massen-, Energie- oder Impulsbilanzierung erben von einer der beiden angesprochenen Bilanzierungsklassen.

Für die Bestimmung des Temperaturfeldes innerhalb einer Wand ist auch eine Elementunterteilung derselben in strömungssenkrechter Richtung vorzunehmen. Die Diskretisierung einer dicken (Rohr-)Wand in radialer Richtung ist in Abbildung 3 veranschaulicht.

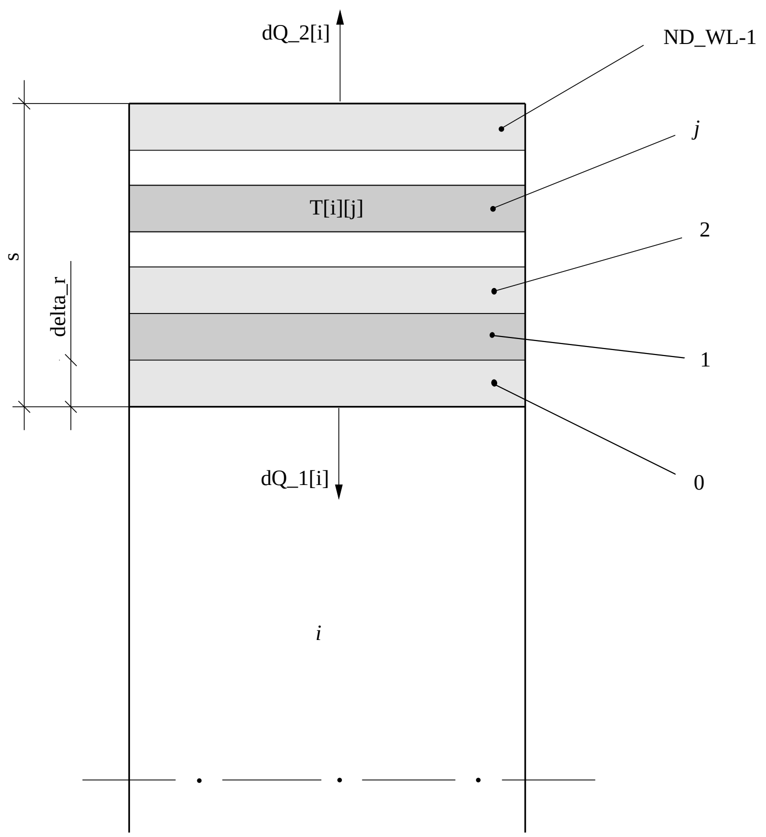

Abbildung 3: Rohrwanddiskretisierung in radialer Richtung

Wird die Wand eines durchströmten Rohres sowohl in Achs- als auch in radialer Richtung diskretisiert, so ist die Temperatur in der Rohrwand doppelt zu indizieren. In den weiteren Ausführungen beschreibt hierbei der erste Index $0 \leq i \leq \mathrm{ND} - 1$ die Diskretisierung in Strömungsrichtung, wohingegen der zweite Index $0 \leq j \leq \mathrm{ND\_WL}-1$ die einzelnen Elemente in rohrwandsenkrechter Richtung unterscheidet.

2.3 Bilanzgleichungen

Der örtliche und zeitliche Zustand eines strömenden Fluids wird durch die instationären Bilanzgleichungen für Masse, Energie und Impuls beschrieben. Je nach Anwendungsfall ist die Verwendung einer, zweier oder aller drei Bilanzierungsgleichungen erforderlich. Ausgehend von den Erhaltungsgleichungen in der allgemeinsten differentiellen Form sollen die Gleichungen vereinfacht, diskretisiert und in SMILE implementiert werden.

2.3.1 Massenbilanzierung

Die Summe der zeitlichen Änderung der Masse im Kontrollvolumen und die durch Strömung zu- und abgeführte Masse ist nach der Kontinuitätsbedingung gleich Null.

Die instationäre Massenbilanz für reine Stoffe lautet im eindimensionalen Fall:

$$\frac{\partial \varrho}{\partial t} + \frac{\partial(\varrho \cdot c)}{\partial x} = 0 \tag{2.9}$$

bzw. eindimensional mit $\dot{m} = \varrho \cdot A_q \cdot c$ unter Annahme konstanter Querschnittsfläche A_q:

$$\frac{\partial \varrho}{\partial t} = -\frac{1}{A_q} \cdot \frac{\partial \dot{m}}{\partial x} \tag{2.10}$$

Nach dem Ort diskretisiert lautet die Massenbilanz dann für $0 \leq i \leq \mathrm{ND} - 1$ mit $V = A_q \cdot \Delta x$

$$\frac{\partial \varrho_i}{\partial t} = \frac{1}{\mathrm{V}} \cdot (\dot{m}_i - \dot{m}_{i+1}) \tag{2.11}$$

Die SMILE-Implementation der Klassen ConcentratedMassBalance, DistributedMassBalance und IncompressibleMassBalance ist in Anhang B.2.1 ab Seite 60 zu finden. Abbildung 18 auf Seite 61 zeigt die Klassenstruktur. Der Tabelle 10 auf Seite 60 können die jeweiligen `@export`-Variablen entnommen werden.

2.3.2 Energiebilanzierung

Die zeitliche Änderung der inneren, kinetischen und potentiellen Energie ist gleich der zu- oder abgeführten Wärme abzüglich der Leistung des Kontrollvolumens gegenüber der Umgebung.

Die instationäre Energiebilanz mit konstantem Strömungsquerschnitt lautet eindimensional in der differentiellen Form:

$$\underbrace{\frac{\partial}{\partial t}\left(\varrho \cdot u + \frac{\varrho \cdot c^2}{2}\right)}_{\text{Speicherung}} + \underbrace{\frac{\partial}{\partial x}\left[c \cdot \left(\varrho \cdot u + \frac{\varrho \cdot c^2}{2}\right)\right]}_{\text{Konvektion}} = \underbrace{\dot{q} \cdot \frac{U}{A}}_{\substack{\text{übertrage-}\\\text{ne Wärme}}} - \underbrace{\frac{\partial}{\partial x}(c \cdot p)}_{\text{Druckfeld}} + \underbrace{\varrho \cdot c \cdot g \cdot \sin\beta}_{\text{Schwerefeld}} \tag{2.12}$$

bzw. mit $u = h - p \cdot v$ in der Enthalpieschreibweise

$$\frac{\partial}{\partial t}\left(\varrho \cdot h + \frac{\varrho \cdot c^2}{2}\right) + \frac{\partial}{\partial x}\left[c \cdot \left(\varrho \cdot h + \frac{\varrho \cdot c^2}{2}\right)\right] = \dot{q} \cdot \frac{U}{A} + \frac{\partial p}{\partial t} + \varrho \cdot c \cdot g \cdot \sin\beta \tag{2.13}$$

β beschreibt den von Rohrachse und Horizontale eingeschlossenen Winkel.

Mit den Vereinfachungen

- $\frac{\varrho \cdot c^2}{2} = 0$ (Mit Enthalpien im Bereich 1000 ... 3500 $\frac{\mathrm{kJ}}{\mathrm{kg}}$ und maximalen Geschwindigkeiten von ca. 30 $\frac{\mathrm{m}}{\mathrm{s}}$ liegt das größtmögliche Verhältnis $\frac{\frac{\varrho \cdot c^2}{2}}{\varrho \cdot h}$ in der Größenordnung $5 \cdot 10^4$.)
- $\varrho \cdot c \cdot g \cdot \sin\beta = 0$ bei waagerechten Rohren
- $\frac{\partial p}{\partial t} = 0$ (Die Massenstrom- und Druckänderungen sind über den Impulssatz eng miteinander verknüpft. Da die Massenstrom- gegenüber der Energieänderung vernachlässigbar gering ist, kann auch davon ausgegangen werden, daß der Druck entsprechend den Verlustmechanismen verzögerungsfrei der Enthalpie bzw. Temperatur folgt.)

lautet die Energiebilanz:

$$\frac{\partial}{\partial t}(\varrho \cdot h) + \frac{\partial}{\partial x}(c \cdot \varrho \cdot h) = \frac{d\dot{Q}}{A_q \cdot dx} \tag{2.14}$$

wegen $\dot{q} = \frac{d\dot{Q}}{U \cdot dx}$. Setzt man in diese Form der Energiebilanz die Massenbilanz (Gleichung (2.9)) ein, so ergibt sich folgende Form:

$$\frac{\partial h}{\partial t} = -c \cdot \frac{\partial h}{\partial x} + \frac{1}{A_q \cdot \varrho} \cdot \frac{d\dot{Q}}{dx} \tag{2.15}$$

Diskretisiert man das Rohr gemäß Abbildung 1 in ND Elemente, so lautet die Energiegleichung für $1 \leq i \leq \mathrm{ND} - 1$ in Enthalpieschreibweise:

$$\frac{\partial h_i}{\partial t} = \frac{1}{\Delta x} \cdot c_i \cdot (h_{i-1} - h_i) + \frac{1}{A_q \cdot \varrho_i} \cdot \frac{\Delta \dot{Q}_i}{\Delta x} \tag{2.16}$$

An den Rohrenden ergeben sich die Randbedingungen

$$\frac{\partial h_0}{\partial t} = \frac{1}{\Delta x} \cdot c_0 \cdot (h_{\mathrm{in}} - h_0) + \frac{1}{A_q \cdot \varrho_0} \cdot \frac{\Delta \dot{Q}_0}{\Delta x} \tag{2.17a}$$

und

$$h_{\mathrm{out}} = h_{\text{ND-1}} \tag{2.17b}$$

bzw. unter Annahme idealen Gases mit $\Delta h = c_p \cdot \Delta T$ in der Temperaturformulierung:

$$\frac{\partial T_i}{\partial t} = \frac{1}{\Delta x} \cdot c_i \cdot (T_{i-1} - T_i) + \frac{1}{A_q \cdot \varrho_i \cdot c_p} \cdot \frac{\Delta \dot{Q}_i}{\Delta x} \tag{2.18}$$

mit den Randbedingungen

$$\frac{\partial T_0}{\partial t} = \frac{1}{\Delta x} \cdot c_0 \cdot (T_{\mathrm{in}} - T_0) + \frac{1}{A_q \cdot \varrho_0 \cdot c_p} \cdot \frac{\Delta \dot{Q}_0}{\Delta x} \tag{2.19a}$$

und

$$T_{\text{out}} = T_{\text{ND-1}} \tag{2.19b}$$

Soll die Energiebilanz für ein im Naturumlauf betriebenes Verdampferrohr aufgestellt werden, so ist in Gleichung 2.12 auch der Schwerefeldterm zu berücksichtigen. Sie lautet dann unter sonst gleichen Vereinfachungen in Enthalpieschreibweise:

$$\frac{\partial h}{\partial t} = -c \cdot \frac{\partial h}{\partial x} + \frac{1}{A_q \cdot \varrho} \cdot \frac{d\dot{Q}}{dx} + c \cdot g \cdot \sin\beta \tag{2.20}$$

Für konstanten Steigungswinkel des Rohres lautet die diskretisierte Gleichung

$$\frac{\partial h_i}{\partial t} = \frac{1}{\Delta x} \cdot c_i \cdot (h_{i-1} - h_i) + \frac{1}{A_q \cdot \varrho_i} \cdot \frac{\Delta \dot{Q}_i}{\Delta x} + c_i \cdot g \cdot \sin\beta \tag{2.21}$$

Die SMILE-Implementation der Klassen ConvectiveTemperatureTransportCM, ConvectiveTemperatureTransportDM, ConvectiveEnthalpyTransportCM und ConvectiveEnthalpyTransportDM ist in Anhang B.2.2 ab Seite 64 zu finden. Abbildung 19 auf Seite 62 zeigt die Klassenstruktur. Der Tabelle 10 auf Seite 60 können die jeweiligen `@export`-Variablen entnommen werden.

2.3.3 Impulsbilanzierung

Für schnell ablaufende Prozesse ist die durch die Impulsbilanz beschriebene Druckdynamik im Dampferzeuger von besonderer Bedeutung. Im Verdampferrohr kann beispielsweise die Impulsbilanz aufgrund starker Beschleunigung des Wassers nicht ohne größere Fehler vernachlässigt werden.

Die zeitliche Änderung des Impulses im Kontrollvolumen ist gleich der Summe aus Impulsänderung durch Zu- und Abstrom, Druckkraft auf die Kontrollvolumenoberfläche, Schwerkraft und Reibungskraft.

Die instationäre Impulsbilanz in x-Richtung lautet unter Annahme konstanten Strömungsquerschnittes und bei Vernachlässigung der diffusiven Terme:

$$\underbrace{\frac{\partial(\varrho \cdot c)}{\partial t}}_{\text{Speicherung}} + \underbrace{\frac{\partial(\varrho \cdot c^2)}{\partial x}}_{\text{Konvektion}} = \underbrace{-\frac{\partial p}{\partial x}}_{\text{Druckfeld}} + \underbrace{\varrho \cdot g \cdot \frac{\partial z}{\partial x}}_{\text{Schwerefeld}} + \underbrace{\left(\frac{\partial p}{\partial x}\right)_{\mathrm{R}}}_{\text{Reibung}} \tag{2.22}$$

Wird der Einfluß des Schwerefelds vernachlässigt, so lautet die Impulsbilanz mit dem in Kapitel 2.6 näher beschriebenen Reibungsdruckabfall $\left(\frac{\partial p}{\partial x}\right)_{\mathrm{R}} = -\frac{\lambda_{\mathrm{R}}}{d} \cdot \varrho \cdot \frac{c \cdot |c|}{2} = -\frac{\lambda_{\mathrm{R}}}{d} \cdot \frac{\dot{m} \cdot |\dot{m}|}{2 \cdot \varrho \cdot A_q^2}$ und mit $\varrho \cdot c = \frac{\dot{m}}{A_q}$

$$\frac{\partial \dot{m}}{\partial t} = A_q \cdot \left(-\frac{\partial p}{\partial x} - \frac{\partial(\varrho \cdot c^2)}{\partial x} - \frac{\lambda_{\mathrm{R}}}{d} \cdot \frac{\dot{m} \cdot |\dot{m}|}{2 \cdot \varrho \cdot A_q^2} \right) \tag{2.23}$$

Geht man wie Schumacher in [52], S. 10, davon aus, daß die örtliche Änderung der kinetischen Energien gegenüber den Änderungen der thermodynamischen Größen vernachlässigbar gering ist, so vereinfacht sich die Gleichung zu

$$\frac{\partial \dot{m}}{\partial t} = A_q \cdot \left(-\frac{\partial p}{\partial x} - \frac{\lambda_{\mathrm{R}}}{d} \cdot \frac{\dot{m} \cdot |\dot{m}|}{2 \cdot \varrho \cdot A_q^2} \right) \tag{2.24}$$

Die Ortsdiskretisierung wird mit Hilfe eines um eine halbe Elementlänge versetzten Kontrollvolumens vorgenommen. Nach dem Aufwärtsdifferenzenverfahren lautet die Gleichung dann mit $\varrho = \varrho_{i-1}$ für $1 \leq i \leq \mathrm{ND} - 1$

$$\frac{\partial \dot{m}_i}{\partial t} = A_q \cdot \left(\frac{p_{i-1} - p_i}{\Delta x} - \frac{\lambda_{\mathrm{R}}}{d \cdot A_q^2} \cdot \frac{\dot{m}_i \cdot |\dot{m}_i|}{2 \cdot \varrho_{i-1}} \right) \tag{2.25}$$

mit den Randbedingungen

$$\dot{m}_0 = \dot{m}_{\mathrm{in}} \tag{2.26a}$$

und

$$\dot{m}_{\mathrm{ND}} = \dot{m}_{\mathrm{out}} \tag{2.26b}$$

bzw. als Differentialgleichungen[2]:

$$\frac{\partial \dot{m}_0}{\partial t} = \frac{1}{\tau} \cdot (\dot{m}_{\mathrm{in}} - \dot{m}_0) \tag{2.26a$'$}$$

und

$$\frac{\partial \dot{m}_{\mathrm{ND}}}{\partial t} = \frac{1}{\tau} \cdot (\dot{m}_{\mathrm{out}} - \dot{m}_{\mathrm{ND}}) \tag{2.26b$'$}$$

Die Geschwindigkeiten können jetzt aus den Massenströmen berechnet werden:

$$c_i = \frac{\dot{m}_i}{A_q \cdot \varrho_i} \qquad \text{für} \quad 0 \leq i \leq \mathrm{ND} - 1 \tag{2.27a}$$

und

$$c_{\mathrm{ND}} = \frac{\dot{m}_{\mathrm{ND}}}{A_q \cdot \varrho_{\mathrm{ND\text{-}1}}} \tag{2.27b}$$

Soll die Impulsbilanz für ein im Naturumlauf betriebenes Verdampferrohr aufgestellt werden, so ist in Gleichung 2.22 auch der Schwerefeldterm zu berücksichtigen. Umgeformt nach der zeitlichen Massenstromänderung lautet die Gleichung damit

$$\frac{\partial \dot{m}}{\partial t} = A_q \cdot \left(-\frac{\partial p}{\partial x} + \varrho \cdot g \cdot \frac{\partial z}{\partial x} - \frac{\lambda_{\mathrm{R}}}{d} \cdot \frac{\dot{m} \cdot |\dot{m}|}{2 \cdot \varrho \cdot A_q^2} \right) \tag{2.28}$$

[2] Unter SMILE müssen alle Komponenten einer vektoriellen Größe mit dem selben Gleichungstyp (hier als Differentialgleichung `@eq diff`) berechnet werden. Die Umformulierung einer algebraischen Gleichung in eine Differentialgleichung wird mit Hilfe des in Kapitel 5.3 näher beschriebenen Integrators vorgenommen.

bzw. ortsdiskretisiert unter Annahme konstanter Rohrsteigung $\frac{\partial z}{\partial x} = \tan\beta$:

$$\frac{\partial \dot{m}_i}{\partial t} = A_q \cdot \left(\frac{p_{i-1} - p_i}{\Delta x} + \varrho_{i-1} \cdot g \cdot \tan\beta - \frac{\lambda_{\mathrm{R}}}{d \cdot A_q^2} \cdot \frac{\dot{m}_i \cdot |\dot{m}_i|}{2 \cdot \varrho_{i-1}} \right) \tag{2.29}$$

Die SMILE-Implementation der Klassen ImpulsbilanzInkompressibel, ImpulsbilanzStationaer und ImpulsbilanzInstationaer ist in Anhang B.2.6 ab Seite 76 zu finden. Abbildung 20 auf Seite 63 zeigt die Klassenstruktur.

2.4 Wärmeübertragung

In den vorangegangenen Kapiteln wurden die beschreibenden Differentialgleichungen der wärmeabgebenden und wärmeaufnehmenden Medien formuliert. Für die weitere mathematische Behandlung werden noch die Wärmeübertragungsbeziehungen benötigt. Es werden die Phänomene Wärmeleitung, konvektiver Wärmeübergang und Wärmestrahlung unterschieden.

2.4.1 Wärmeleitung

Soll das Temperatur-Zeitverhalten von Wänden beschrieben werden, so ist das in diesem Kapitel dargestellte Modell Waermeleitung oder das in Kapitel 2.5 erläuterte vereinfachte Modell DiscreteHeatStore zu verwenden. Ist das Temperaturfeld in der Wand von Interesse, z. B. zur Bestimmung von Wärmespannungen in dickwandigen Bauteilen, so muß die Wand diskretisiert werden, und der sonst häufig vernachlässigbare Einfluß der Wärmeleitung ist zu berücksichtigen.

Die Energiebilanz für ein Volumenelement eines ruhenden, inkompressiblen Mediums lautet nach [43]

$$\varrho \cdot c_p \cdot \frac{\partial T}{\partial t} = -\nabla \dot{q} + W \tag{2.30}$$

Hierbei bezeichnen T die Temperatur, ϱ die Dichte, c_p die spezifische Wärmekapazität bei konstantem Druck, $\dot{q}$ die Wärmestromdichte und $W = W(\mathrm{Ort}, \mathrm{Zeit})$ die von einer Wärmequelle erzeugte Wärmemenge je Raum- und Zeiteinheit. Die Wärmestromdichte ergibt sich über die Wärmeleitfähigkeit λ aus dem Gradienten des Temperaturfelds:

$$\dot{q} = -\lambda \cdot \nabla T \tag{2.31}$$

Die Kopplung von Gleichung (2.30) und (2.31) liefert unter Annahme konstanter Wärmeleitfähigkeit λ:

$$\frac{\partial T}{\partial t} = \frac{\lambda}{c_p \cdot \varrho} \cdot \nabla^2 T + \frac{1}{c_p \cdot \varrho} \cdot W \tag{2.32}$$

Im folgenden soll die Wärmeleitung in Rohrwandungen untersucht werden. Wird der Einfluß der Wärmeleitung in Achsrichtung vernachlässigt, und ist die Wärmeleitung radialsymmetrisch, so reicht es aus, das Phänomen eindimensional zu betrachten. Da i. a. die Wandstärke deutlich kleiner ist als der Rohrdurchmesser, kann die Rohrkrümmung vernachlässigt und das Problem auf das einer ebenen Wand reduziert werden. Mit der Temperaturleitfähigkeit $a = \frac{\lambda}{c_p \cdot \varrho}$ und der erzeugten massenspezifischen Wärmeleistung $W = 0$ lautet die Gleichung unter den getroffenen

Annahmen in karthesischen Koordinaten:

$$\frac{\partial T}{\partial t} = a \cdot \frac{\partial^2 T}{\partial x^2} \tag{2.33}$$

Sollen hingegen dickwandige Rohre oder Dampftrommeln modelliert werden, so ist Gleichung (2.32) in Polarkoordinaten zu schreiben. Unter Annahme von Radialsymmetrie lautet die Fouriersche Differentialgleichung der Wärmeleitung dann

$$\frac{\partial T}{\partial t} = a \cdot \frac{1}{r} \cdot \frac{\partial}{\partial r}\left(r \cdot \frac{\partial T}{\partial r}\right) \tag{2.34}$$

Diskretisiert man die Wand gemäß Abbildung 3 in ND_WL Elemente, so lautet die Gleichung in karthesischen Koordinaten für $1 \leq j \leq \mathrm{ND\_WL}-2$:

$$\frac{dT_j}{dt} = \frac{a}{(\Delta x)^2} \cdot (T_{j+1} + T_{j-1} - 2 \cdot T_j) \tag{2.35}$$

Die Randbedingungen (hier gegeben über die Wärmeleistungen an Rohrinnen- und -außenseite)

$$T_0 = T_1 - \frac{\Delta x}{\lambda} \cdot \dot{q}_1 \tag{2.36a}$$

und

$$T_{\mathrm{ND\_WL}-1} = T_{\mathrm{ND\_WL}-2} - \frac{\Delta x}{\lambda} \cdot \dot{q}_2 \tag{2.36b}$$

müssen (aus bereits in Kapitel 2.3.3 erläuterten Gründen) über künstliche Integratoren und Hilfszeitkonstanten als Differentialgleichungen geschrieben werden:

$$\frac{dT_0}{dt} = \frac{1}{\tau} \cdot \left(T_1 - T_0 - \frac{\Delta x}{\lambda} \cdot \dot{q}_1\right) \tag{2.36a$'$}$$

und

$$\frac{dT_{\mathrm{ND\_WL}-1}}{dt} = \frac{1}{\tau} \cdot \left(T_{\mathrm{ND\_WL}-2} - T_{\mathrm{ND\_WL}-1} - \frac{\Delta x}{\lambda} \cdot \dot{q}_2\right) \tag{2.36b$'$}$$

Die flächenspezifische Wärmeleistung $\dot{q}_i$ an der Wandinnen- und -außenfläche kann jeweils aus dem Verhältnis der Wärmeleistung und der Oberfläche des Diskretisierungselements bestimmt werden. Wenn A die Oberfläche der gesamten Wand bezeichnet, so gilt

$$\dot{q}_i = \frac{dQ_i}{A/\mathrm{ND}} \tag{2.37}$$

Um das Modell zu validieren, soll die analytische Lösung eines speziellen Problems mit der entsprechenden numerischen Lösung verglichen werden: Ein halbunendlicher Körper wird einseitig mit einer schwellenden Temperatur beaufschlagt (z. B. täglicher oder jährlicher Wechsel der Sonneneinstrahlung auf Gebäude oder Wärmeübertragung in Regeneratoren) — die Randbedingung der beaufschlagten Seite lautet jetzt

$$T_0 = \overline{T} + \widehat{T} \cdot \cos(\omega \cdot t) \tag{2.38}$$

bzw. als Differentialgleichung

$$\frac{dT_0}{dt} = \frac{1}{\tau} \cdot \left(\overline{T} + \widehat{T} \cdot \cos(\omega \cdot t) - T_0 \right) \tag{2.38'}$$

Bezeichnet x die Entfernung von der mit der Schwelltemperatur beaufschlagten Fläche, so lautet die analytische Lösung für die normierte Körpertemperatur Θ mit dem Temperaturmittelwert $\overline{T}$ und der Schwellamplitude $\widehat{T}$

$$\Theta = \frac{T - \overline{T}}{\widehat{T}} = \exp\left(-\frac{1}{\sqrt{2}} \cdot \sqrt{\frac{\omega}{a}} \cdot x \right) \cdot \cos\left(-\frac{1}{\sqrt{2}} \cdot \sqrt{\frac{\omega}{a}} \cdot x + \omega \cdot t \right) \tag{2.39}$$

In Abbildung 4 ist der Verlauf der normierten Temperatur über den Ort für einen festen Zeitpunkt grafisch dargestellt. Hierbei wird die analytische mit der numerischen Lösung für verschiedene Hilfszeitkonstanten und unterschiedliche Anzahlen von Diskretisierungselementen verglichen. Es zeigt sich, daß eine Zeitkonstante τ sinnvolle Ergebnisse liefert, wenn sie in der

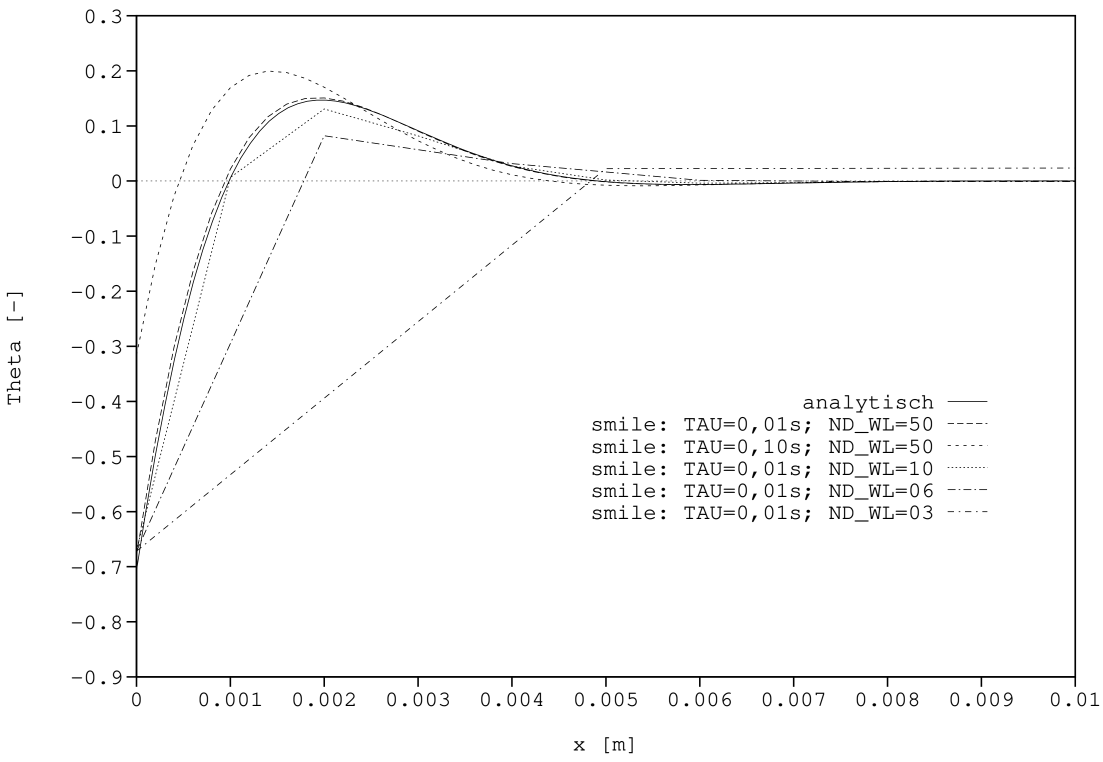

Abbildung 4: Vergleich instationäre Wärmeleitung, analytische und numerische Lösung

Größenordnung $\tau = 1\% \cdot \frac{2\pi}{\omega}$ gewählt wird.

Die SMILE-Implementation der Klassen Waermeleitung und WaermeleitungRohr ist in Anhang B.4.1 ab Seite 99 zu finden. Abbildung 22 auf Seite 98 zeigt die Klassenstruktur. Der Tabelle 13 auf Seite 98 können die jeweiligen `@export`-Variablen entnommen werden.

2.4.2 Konvektion

Die Bestimmung des Wärmeübergangskoeffizienten α bei Wärmeübertragung durch Konvektion kann für verschiedene Wärmeübergangsphänomene zum Teil sehr aufwendig sein. Es werden in der Literatur viele Bestimmungsgleichungen vorgeschlagen, die rechenzeitintensive Funktionen wie log, pow und sqrt enthalten. Der Aufwand zur α-Berechnung sollte jedoch immer in einem sinnvollen Verhältnis zum Gesamtaufwand stehen. Es kann in Einzelfällen durchaus zweckmäßig sein, ein Modell „konstanter Wärmeübergangskoeffizient“ oder gar „konstanter Wärmedurchgangskoeffizient“ zu verwenden. Die Austauschbarkeit von Modellen in der objektorientierten Programmierung liefert ein ausgeprägtes Mittel, die Modellierungstiefe der Einzelkomponenten dem zu simulierenden Gesamtsystem anzupassen.

Die durch Konvektion übertragene Wärmeleistung berechnet sich nach

$$dQ_i = \alpha \cdot \frac{f \cdot A}{\mathrm{ND}} \cdot (T_{\mathrm{wall},i} - T_{\mathrm{fluid},i}) \tag{2.40}$$

Hierbei bezeichnen α den Wärmeübergangskoeffizienten, $\frac{A}{\mathrm{ND}}$ die Wärmeübertragungsfläche pro Diskretisierungselement und $T_{\mathrm{wall}} - T_{\mathrm{fluid}}$ die treibende Temperaturdifferenz. Die Größe f stellt einen Flächenfaktor dar, der eine Flächenvergrößerung bzw. -verkleinerung, z. B. bei berippten Rohren, Rohrverschmutzung oder nur einseitiger Rohrbeaufschlagung berücksichtigen kann.

Der Wärmeübergangskoeffizient kann aus der dimensionslosen Nusselt-Zahl Nu bestimmt werden:

$$\alpha = \frac{\mathrm{Nu} \cdot \lambda}{L_0} \tag{2.41}$$

Je nach Strömungssituation sind unterschiedliche Gleichungen für die Nusselt-Zahl Nu gegeben. Sie ist i. a. abhängig von zwei weiteren dimensionslosen Größen: von der Reynolds-Zahl

$$\mathrm{Re} = \frac{|c| \cdot L_0 \cdot \rho}{\eta} \tag{2.42}$$

und von der Prandtl-Zahl

$$\mathrm{Pr} = \frac{\eta \cdot c_p}{\lambda} \tag{2.43}$$

Die SMILE-Implementation der Wärmeübergangsklassen ist in Anhang B.3.1 ab Seite 83 zu finden. Abbildung 21 auf Seite 84 zeigt die Klassenstruktur. Der Tabelle 11 auf Seite 85 können die jeweiligen `@export`-Variablen entnommen werden.

2.4.2.1 Wärmeübergang auf der Rauchgasseite Wird die Reynolds-Zahl mit der mittleren Geschwindigkeit im Hohlraum einer Rohrreihe und dem halben Umfang eines Einzelrohres („Überströmlänge“) als charakteristische Länge gebildet, so kann die Nusselt-Zahl des gesamten umströmten Rohrbündels nach [27], Kapitel Gf, über einen Anordnungsfaktor f_A aus der Nusselt-Zahl eines umströmten Einzelrohres ermittelt werden:

$$\mathrm{Nu}_{\mathrm{Rb}} = f_A \cdot \mathrm{Nu}_1 \tag{2.44}$$

Die relevante Reynolds-Zahl ergibt sich als Quotient der Reynolds-Zahl aus Gleichung (2.42) und dem Hohlraumanteil ψ:

$$\mathrm{Re}_{\psi,1} = \frac{\mathrm{Re}}{\psi} \tag{2.45}$$

Berechnet man eine laminare

$$\mathrm{Nu}_{1,\mathrm{lam}} = 0,664 \cdot \sqrt{\mathrm{Re}_{\psi,1}} \cdot \sqrt[3]{\mathrm{Pr}} \tag{2.46}$$

und eine turbulente Nusseltzahl für ein Einzelrohr

$$\mathrm{Nu}_{1,\mathrm{turb}} = \frac{0,037 \cdot \mathrm{Re}_{\psi,1}^{0,8} \cdot \mathrm{Pr}}{1 + 2,443 \cdot \mathrm{Re}_{\psi,1}^{-0,1} \cdot (\mathrm{Pr}^{2/3} - 1)} \tag{2.47}$$

so ergibt sich die Nusselt-Zahl des Einzelrohres zu

$$\mathrm{Nu}_1 = 0,3 + \sqrt{\mathrm{Nu}_{1,\mathrm{lam}}^2 + \mathrm{Nu}_{1,\mathrm{turb}}^2} \tag{2.48}$$

Anordnungsfaktor f_A und Hohlraumanteil ψ für ein Rohrbündel können hierfür in [27], Kapitel Gf, nachgeschlagen werden.

2.4.2.2 Wärmeübergang auf der Arbeitsstoffseite Auf der Wasserseite sind Wärmeübergänge ohne und mit Phasenwechsel zu unterscheiden.

2.4.2.2.1 Wärmeübergang ohne Phasenwechsel Je nach Rohrströmungsform sind für die Bestimmung des Wärmeübergangskoeffizienten α verschiedene Gleichungen zur Berechnung der Nusselt-Zahl heranzuziehen. Ist die Strömung hydrodynamisch ausgebildet laminar ($\mathrm{Re} \leq 2300$), so kann nach [27] Gb mit einem konstanten Wert von

$$\mathrm{Nu}_{\mathrm{lam}} = 3,66 \tag{2.49}$$

unter der Annahme konstanter Wandtemperatur bzw. mit

$$\mathrm{Nu}_{\mathrm{lam}} = 4,364 \tag{2.50}$$

für konstante Wärmestromdichte gerechnet werden. Bei turbulenter Strömung ($10^4 < \mathrm{Re}$) kann die Nusselt-Zahl nach der Gleichung von Gnielinski bestimmt werden:

$$\mathrm{Nu}_{\mathrm{turb}} = \frac{(\mathrm{Re} - 1000,0) \cdot \mathrm{Pr} \cdot \xi/8}{1 + 12,7 \cdot \sqrt{\xi/8} \cdot (\mathrm{Pr}^{2/3} - 1)} \cdot \left(1 + (d_i/l)^{2/3}\right) \tag{2.51}$$

mit dem Druckverlustbeiwert nach Filonenko

$$\xi = \frac{1}{(1,82 \cdot \log_{10}(\mathrm{Re}) - 1,64)^2} \tag{2.52}$$

Der die Einlaufströmung berücksichtigende Korrekturterm $1 + (d_i/l)^{2/3}$ kann bei langen Rohren entfallen.

Die Gleichung besitzt zwar auch Gültigkeit für den Übergangsbereich ($2300 < \mathrm{Re} \leq 10^4$), trotzdem sollte aber, um einen stetigen Verlauf des Wärmeübergangskoeffizienten über die Reynolds-Zahl zu erhalten, im Übergangsbereich mit dem zwischen $\mathrm{Nu}_{\mathrm{lam}}$ und $\mathrm{Nu}_{\mathrm{turb}}$ linear gemittelten Wert gerechnet werden:

$$\mathrm{Nu}_{\ddot{\mathrm{U}}\mathrm{g}} = \mathrm{Nu}_{\mathrm{lam}} \cdot \frac{10^4 - \mathrm{Re}}{10^4 - 2300} + \mathrm{Nu}_{\mathrm{turb}} \cdot \frac{\mathrm{Re} - 2300}{10^4 - 2300} \tag{2.53}$$

Für Überschlagsrechnungen werden häufig einfachere Beziehungen als Gleichung (2.51) benötigt.

Von Gnielinski werden beispielsweise in [27] die Bestimmungsgleichungen

$$\mathrm{Nu} = 0,0214 \cdot \left(\mathrm{Re}^{0,8} - 100\right) \cdot \mathrm{Pr}^{0,4} \cdot \left(1 + (d_i/l)^{2/3}\right) \tag{2.54}$$

für den Bereich $0,5 < \mathrm{Pr} < 1,5$ und

$$\mathrm{Nu} = 0,012 \cdot \left(\mathrm{Re}^{0,87} - 280\right) \cdot \mathrm{Pr}^{0,4} \cdot \left(1 + (d_i/l)^{2/3}\right) \tag{2.55}$$

für $1,5 < \mathrm{Pr} < 500$ vorgeschlagen.

2.4.2.2.2 Wärmeübergang mit Phasenwechsel

Verdampfung Soll der Wärmeübergangskoeffizient für Verdampfung von Wasser bestimmt werden, so sind mit steigender Temperaturdifferenz zwischen Wand und Fluid $\vartheta_{\mathrm{W}} - \vartheta_{\mathrm{Fl}}$ vier Phänomene zu unterscheiden:

- freie Konvektion,
- Blasensieden,
- stabile Filmverdampfung und
- instabile Filmverdampfung.

Theoretisch könnte in SMILE nach den verschiedenen Verdampfungsphänomenen differenziert und entsprechend unterschiedliche Gleichungssätze zur Verfügung gestellt werden. Da das Modell jedoch unnötig kompliziert werden würde, wird hier bei Verdampfung grundsätzlich von Blasensieden ausgegangen. Dies ist das in technischen Verdampfern bedeutendste Verdampfungsphänomen — Filmsieden tritt nach [32] bei genügend großem Verhältnis $\frac{\dot{m}_A}{\dot{q}_A}$ nicht auf.

Da die Physik äußerst kompliziert ist, werden in der Literatur unterschiedliche empirische Bestimmungsgleichungen für den Wärmeübergangskoeffizienten in Abhängigkeit vom Druck p und von der Wärmestromdichte $\dot{q}$ angegeben. Nach [26] kann für den Druckbereich von etwa $0,1\ \mathrm{bar} \leq p \leq 150\ \mathrm{bar}$ folgende Zahlenwertgleichung angegeben werden:

$$\alpha = 1,86 \cdot \dot{q}^{0,72} \cdot p^{0,24} \tag{2.56}$$

mit α in $\frac{\mathrm{kcal}}{\mathrm{m^2\,h\,grd}}$, $\dot{q}$ in $\frac{\mathrm{kcal}}{\mathrm{m^2 h}}$ und p in at bzw.

$$\alpha = 1,95 \cdot \dot{q}^{0,72} \cdot p^{0,24} \tag{2.57}$$

mit α in $\frac{\mathrm{W}}{\mathrm{m^2K}}$, $\dot{q}$ in $\frac{\mathrm{W}}{\mathrm{m^2}}$ und p in bar.

Um in der Simulation eine „algebraische Schleife“ (siehe hierzu auch Kapitel 5.3), $\dot{q} = f(\alpha)$ und $\alpha = f(\dot{q})$, zu vermeiden, ist es sinnvoll, die Gleichung für die Wärmestromdichte

$$\dot{q} = \alpha \cdot (T_{\mathrm{wall}} - T_{\mathrm{fluid}}) \tag{2.58}$$

direkt in Gleichung (2.57) einzusetzen. Man erhält nunmehr den Wärmeübergangskoeffizienten in Abhängigkeit von der Temperaturdifferenz und dem Druck:

$$\alpha = 1,95^{\frac{1}{1-0,72}} \cdot (T_{\mathrm{wall}} - T_{\mathrm{fluid}})^{\frac{0,72}{1-0,72}} \cdot p^{\frac{0,24}{1-0,72}} = 10,86 \cdot (T_{\mathrm{wall}} - T_{\mathrm{fluid}})^{2,57} \cdot p^{0,86} \tag{2.59}$$

Eine etwas aufwendigere Gleichung zur Berechnung des Wärmeübergangskoeffizienten bei Behältersieden wird in [28] vorgeschlagen:

$$\frac{\alpha}{\alpha_0} = C_W \cdot F(p^*) \cdot \left(\frac{\dot{q}}{\dot{q}_0}\right)^n \tag{2.60}$$

Hierbei bezeichnen $\frac{\alpha}{\alpha_0}$ den normierten Wärmeübergangskoeffizienten, C_W einen Faktor zur Beschreibung des relativen Einflusses der Heizflächeneigenschaften,

$$F(p^*) = 1,73 \cdot (p^*)^{0,27} + \left(6,1 + \frac{0,68}{1-p^*}\right) \cdot (p^*)^2 \tag{2.61}$$

eine Funktion des auf den kritischen Druck bezogenen Druckes $p^* = \frac{p}{p_{\mathrm{krit}}}$, $\frac{\dot{q}}{\dot{q}_0}$ die normierte Wärmestromdichte und

$$n = 0,9 - 0,3 \cdot (p^*)^{0,15} \tag{2.62}$$

einen druckabhängigen Exponenten. Der Normierungszustand ist frei wählbar, z. B. $\dot{q}_0 = 20\,000$ $\frac{\mathrm{W}}{\mathrm{m}^2}$ und $p_0^* = 0,1$. Der Wert für α_0 muß aus Messungen oder rechnerisch — unabhängig von Gleichung (2.60) — bestimmt werden.

Um Unstetigkeiten an den Phasengrenzen zu vermeiden, sollte nach [46] im grenznahen Naßdampfbereich zwischen dem ein- und dem zweiphasigen Wärmeübergangskoeffizienten interpoliert werden: Liegt der Dampfmassenstromanteil

$$\dot{x} = \frac{\dot{m}''}{\dot{m}' + \dot{m}''} \tag{2.63}$$

im Bereich $0,0 < \dot{x} < 0,1$, so gilt:

$$\alpha = (1 - 10 \cdot \dot{x}) \cdot \alpha_{1\mathrm{Ph}} + 10 \cdot \dot{x} \cdot \alpha_{2\mathrm{Ph}} \tag{2.64a}$$

Für den Dampfmassenstromanteil im Bereich $0,9 < \dot{x} < 1,0$ gilt:

$$\alpha = (10 \cdot \dot{x} - 9) \cdot \alpha_{1\mathrm{Ph}} + (10 - 10 \cdot \dot{x}) \cdot \alpha_{2\mathrm{Ph}} \tag{2.64b}$$

Kondensation Kondensation in Dampferzeugern ist z. B. von Bedeutung bei Aufheizung von noch kalten Überhitzerbauteilen. Es kann in guter Näherung mit dem konstanten Wärmeübergangskoeffizienten

$$\alpha = 8000\ \frac{\mathrm{W}}{\mathrm{m}^2\mathrm{K}} \tag{2.65}$$

gerechnet werden, da i. a. der Einfluß auf der Kühlseite den Wärmedurchgang aufgrund des größeren Wärmeübergangswiderstandes dominiert.

2.4.3 Strahlung

Bei hohen Temperaturen ist der Anteil der berührungslos übertragenen Wärme besonders groß. Die Strahlungswärmeleistung berechnet sich nach dem „Gesetz von Stefan-Boltzmann“ mit Temperaturen in K zu

$$dQ_i = \varepsilon \cdot C_S \cdot \frac{A}{\mathrm{ND}} \cdot (T^4_{\mathrm{wall},i} - T^4_{\mathrm{strlg},i}) \tag{2.66}$$

Die Emissionszahl ε ergibt sich je nach Geometrie der beiden Strahlungspartner aus deren Emissionsverhältnissen ε_1 und ε_2. Im Rahmen der Gesamtgenauigkeit eines zu simulierenden Systems kann i. a. die Emissionszahl konstant angesetzt werden.

Soll neben der Wärmeübertragung durch Strahlung auch konvektiver Wärmeübergang berücksichtigt werden, so können die jeweiligen Wärmeleistungsanteile — aufgrund der meist nahezu zutreffenden Annahme der Unabhängigkeit beider Teilphänomene — additiv überlagert werden:

$$dQ_i = dQ_{\mathrm{strlg},i} + dQ_{\mathrm{konv},i} = \varepsilon \cdot C_S \cdot \frac{A}{\mathrm{ND}} \cdot (T^4_{\mathrm{wall},i} - T^4_{\mathrm{strlg},i}) + \alpha \cdot \frac{A}{\mathrm{ND}} \cdot (T_{\mathrm{wall},i} - T_{\mathrm{fluid},i}) \tag{2.67}$$

Es gilt $T_{\mathrm{strlg},i} = T_{\mathrm{fluid},i}$. Wie groß der Anteil der durch Strahlung übertragenen Wärme bezogen auf die konvektiv übertragene Wärmeleistung ist, zeigt für einen mittleren Wert für den Wärmeübergangskoeffizienten $\alpha = 50\frac{\mathrm{W}}{\mathrm{m^2K}}$ und einen Emissionskoeffizienten $\varepsilon = 0,80$ (nach [62], S. Ka3; Stahl, dicke rauhe Oxidschicht) beispielhaft die Abbildung 5. Aufgetragen ist das Wärmeleistungsverhältnis

$$\frac{dQ_{\mathrm{strlg}}}{dQ_{\mathrm{konv}}} = \frac{\varepsilon \cdot C_S}{\alpha} \cdot \frac{T^4_{\mathrm{fluid}} - T^4_{\mathrm{wall}}}{T_{\mathrm{fluid}} - T_{\mathrm{wall}}} \tag{2.68}$$

über verschiedene Temperaturdifferenzen $\Delta T = T_{\mathrm{fluid}} - T_{\mathrm{wall}}$ und unterschiedliche Wandtemperaturen T_{wall}. Liegt der Quotient deutlich über 1,0, so kann der Konvektionseinfluß vernachlässigt werden.

2.4.4 Wärmedurchgang

Ist statt zweier Wärmeübergangs- α und eines Wärmeleitkoeffizienten λ der Wärmedurchgangskoeffizient k bekannt, so kann die Wärmeleistung einfach berechnet werden nach

$$dQ_i = k \cdot \frac{A}{\mathrm{ND}} \cdot (T_{\mathrm{fluid1},i} - T_{\mathrm{fluid2},i}) \tag{2.69}$$

Ein Modell WaermedurchgangConstK berücksichtigt nicht die Wärmespeicherung in der Trennwand. Soll das dynamische Verhalten dennoch simuliert werden, so ist die Verknüpfung zweier Wärmeübergangsmodelle mit einem Wärmespeicher- bzw. -leitmodell unumgänglich.

Die SMILE-Implementation der Klassen WaermedurchgangConstK, WaermedurchgangRohrWueKonvHeatStoreWueKonv und WaermedurchgangRohrWueKonvWaermeleitungWueKonv ist in Anhang B.3.3 ab Seite 95 zu finden. Der Tabelle 12 auf Seite 96 können die jeweiligen `@export`-Variablen entnommen werden.

2.5 Wärmespeicherung

Zur Modellierung der Trennwand zwischen zwei Fluiden kann neben dem Modell Waermeleitung auch ein einfacheres Wärmespeichermodell DiscreteHeatStore verwendet werden. Es geht davon aus, daß in der Wand an jeder Stelle die gleiche Temperatur vorliegt — die Wand muß hier also nicht radial diskretisiert werden. Diese Annahme ist sinnvoll für dünne Wände, also z. B. für

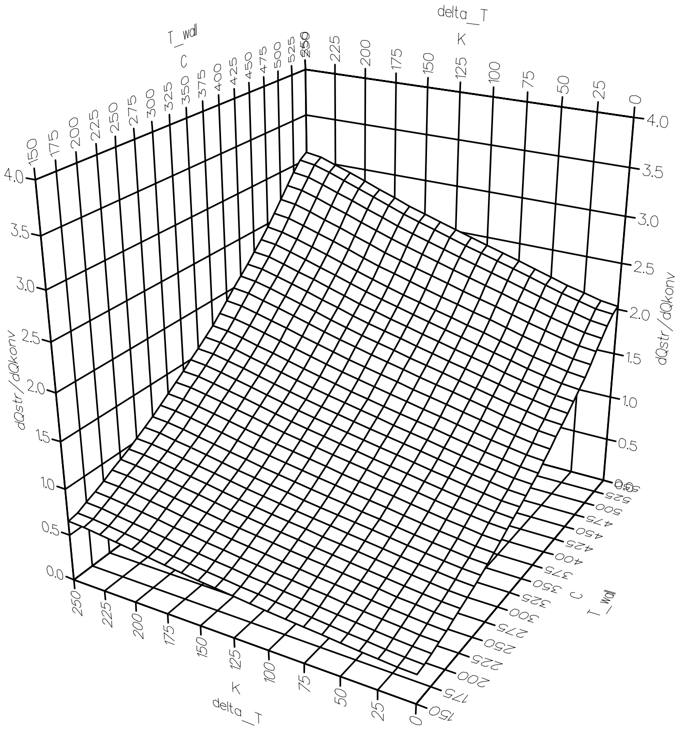

Abbildung 5: Verhältnis $\frac{dQ_{\mathrm{strlg}}}{dQ_{\mathrm{konv}}}$ für $\alpha = 50 \frac{\mathrm{W}}{\mathrm{m}^2\mathrm{K}}$ und $\varepsilon = 0,80$

Rohre mit $d_\mathrm{m} \gg s$.

Die Wandtemperatur verändert sich in Abhängigkeit von der Wärmekapazität der Wand und der Summe der ein- und austretenden Wärmeströme:

$$\frac{\partial T}{\partial t} = \frac{\dot{Q}}{m \cdot c_p} \tag{2.70}$$

bzw. unter Berücksichtigung der Rohrdiskretisierung in Achsrichtung:

$$\frac{\partial T_i}{\partial t} = \frac{\dot{Q}_i}{m \cdot c_p} \tag{2.71}$$

Werden wie bei Cellier ([13]) „Through"- von „Across"-Variablen unterschieden[3], so ist $\dot{Q}_i$ eine „Through"-Größe, und damit richtungs- und vorzeichenbehaftet. Wird ein Wärmestrom positiv gezählt, wenn er von der Wand in Richtung des Fluids fließt, und bezeichnen 1 und 2 die beiden Wandseiten, so ergibt sich die wandtemperaturverändernde Summe der Wärmeleistungen eines Diskretisierungselementes zu

$$\dot{Q}_i = -(\dot{Q}_{1,i} + \dot{Q}_{2,i}) \tag{2.72}$$

Die SMILE-Implementation der Wärmespeicherklassen DiscreteHeatStore, ThinWall und ThinWallTubes ist in Anhang B.4.1 ab Seite 99 zu finden. Abbildung 22 auf Seite 98 zeigt die Klassenstruktur. Der Tabelle 13 auf Seite 98 können die jeweiligen `@export`-Variablen entnommen werden.

2.6 Reibungsdruckabfall

Soll der Reibungsdruckabfall in einem Dampferzeugerrohr berücksichtigt werden, so gibt es zwei Möglichkeiten, dies zu realisieren: Einerseits wird der Druckverlust als an einem Rohrende konzentriert betrachtet, er muß also für ein Rohr pro Zeitschritt nur einmal berechnet werden. Andererseits wird der Druckabfall als kontinuierlich über die Rohrlänge verteilt angesehen. Dies beschreibt die Realität zwar genauer, impliziert aber einen höheren Rechenaufwand. Ist eine Impulsbilanzierung der einzelnen Kontrollvolumen eines diskretisierten Dampferzeugerrohres bei kompressiblem Arbeitsmittel vorgesehen, so muß mit einem kontinuierlichen Druckverlauf gerechnet werden.

Der Druckverlustterm der Impulsbilanz (Gleichung (2.22)) berechnet sich nach [20], S. 23, für das Einphasengebiet zu

$$\left(\frac{\partial p}{\partial x}\right)_\mathrm{R} = \left(\frac{\lambda_\mathrm{R}}{d_h} + \frac{d\xi}{dx}\right) \cdot \varrho \cdot \frac{\mathrm{c}^2}{2} \tag{2.73}$$

$d_h = \frac{4 \cdot A_q}{U}$ bezeichnet hierbei den hydraulischen Durchmesser. Er entspricht im Fall von Rohren

[3] Cellier erläutert die Begriffe an einem elektrischen Netzwerk. Er führt Zählpfeile ein und versteht unter „Through"-Größen solche Variablen, die an einem Schnittpunkt aufsummiert Null ergeben müssen. Der 1. Kirchhoffsche Satz besagt beispielsweise, daß in jedem Knoten die Summe der Ströme Null ist. Variablen, die an einem gemeinsamen Punkt gleichzusetzen sind, bezeichnet er als „Across"-Variablen. Hierzu zählt beim elektrischen Netzwerk z. B. das elektrische Potential.

mit Kreisquerschnitt dem Innendurchmesser d_i. In dieser Druckverlustbeziehung erfaßt λ_{R} die reine Rohrreibung, während ξ nicht erfaßte Verlustmechanismen beschreibt. Geht man von reiner Rohrreibung runder Rohre aus, so ergibt sich der Druckverlust pro Diskretisierungselement wegen $d_h = d_i$ und $\xi = 0$ zu

$$\Delta p = \lambda_R \cdot \frac{\Delta x}{d_i} \cdot \varrho \cdot \frac{\mathrm{c}^2}{2} \tag{2.74}$$

bzw. umgeformt nach $c = c_i$ mit $\varrho = \frac{\varrho_{i-1}+\varrho_i}{2}$ und $\Delta p = p_{i-1} - p_i$ für $1 \leq i \leq ND - 1$:

$$c_i = \sqrt{\frac{4 \cdot (p_{i-1} - p_i) \cdot d_i}{\lambda_R \cdot (\varrho_{i-1} + \varrho_i) \cdot \Delta x}} \tag{2.75}$$

mit den Randbedingungen

$$c_0 = \frac{\dot{m}_{\mathrm{in}}}{A_q \cdot \varrho_{\mathrm{in}}} \tag{2.76a}$$

und

$$c_{\mathrm{ND}} = \frac{\dot{m}_{\mathrm{out}}}{A_q \cdot \varrho_{\mathrm{ND\text{-}1}}} \tag{2.76b}$$

Damit die Funktion $c = f(\Delta p)$ überall stetig und stetig differenzierbar ist, ist es sinnvoll, für SMILE die Funktion zu manipulieren und in Teilfunktionen zu untergliedern. Mit dem Radikanden $y = \frac{4 \cdot \Delta p \cdot d_i}{\lambda_R \cdot (\varrho_{i-1}+\varrho_i) \cdot \Delta x}$ bietet sich an:

$$c_i = \begin{cases} -\sqrt{-y} & & y < -1,0 \\ -\frac{1}{4}y^3 + \frac{5}{4}y & \text{für} & -1,0 \leq y \leq 1,0 \\ \sqrt{y} & & 1,0 < y \end{cases} \tag{2.77}$$

Der (manipulierte) Funktionsverlauf $c = f(y)$ ist grafisch in Abbildung 6 dargestellt.

Eine Vereinfachung der Beziehung $c = f(\Delta p)$ wird erreicht, wenn lineare Abhängigkeit angenommen wird: $c \sim \Delta p$. Ausgehend von in etwa bekannter Geschwindigkeit und Druckverlust im stationären (oder einem beliebigen anderen) Zustand kann die Geschwindigkeit dann nach folgender Gleichung ermittelt werden:

$$c_i = c_{\mathrm{stat}} \cdot \frac{p_{i-1} - p_i}{(\Delta p)_{\mathrm{stat}}} \tag{2.78}$$

Soll der Druckverlust im Zweiphasengebiet bestimmt werden, so beschreibt der „Zweiphasenmultiplikator“ Φ_{2ph} nach [46], S. 54, das Verhältnis zwischen berechneten Druckverlustwerten im Zwei- und im Einphasengebiet:

$$\Delta p_{\mathrm{R,2ph}} = \Phi_{\mathrm{2ph}} \cdot \Delta p_{\mathrm{R,1ph}} \tag{2.79}$$

Er berechnet sich aus dem Dampfmassenstromanteil $\dot{x}$ und den Dichten im Sättigungszustand für flüssige (ϱ') und dampfförmige (ϱ'') Phase:

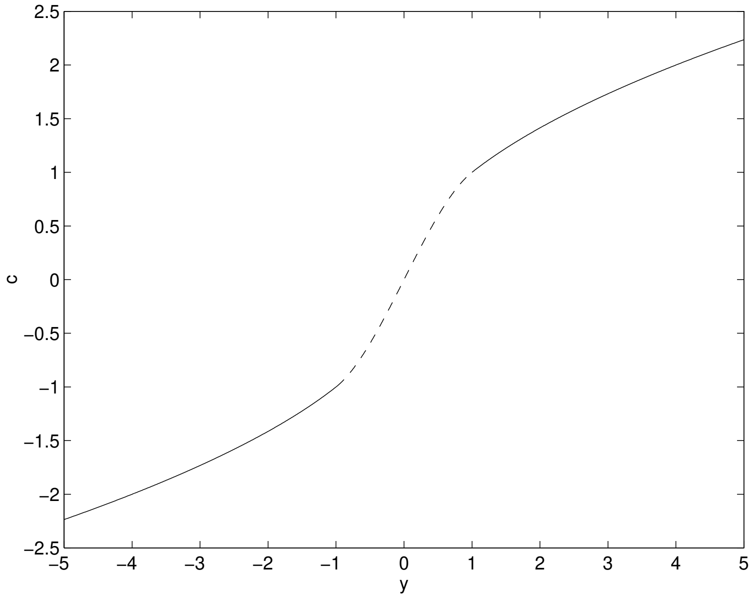

Abbildung 6: Berechnung von c in Abhängigkeit vom Radikanden y

$$\Phi_{\mathrm{2ph}} = 1 + \dot{x} \cdot \left(\frac{\varrho'}{\varrho''} - 1 \right) \tag{2.80}$$

Eine andere, experimentell ermittelte Gleichung zur Bestimmung des Zweiphasenmultiplikators für den Bereich $0,1 \leq p^* \leq 0,9$ wird in [20], S. 56, angegeben — sie vermeidet weitere Stoffwertfunktionsaufrufe:

$$\Phi_{\mathrm{2ph}} = 1 + \frac{1 - p^*}{p^*} \cdot F(p^*) \cdot \dot{x} \cdot \left(1 + 1,5 \cdot (1 - \dot{x})^{0,36} \right) \tag{2.81}$$

mit der Druckfunktion

$$F(p^*) = 4,75 + 9,03 \cdot p^* - 21,25 \cdot (p^*)^2 + 24,01 \cdot (p^*)^3 \tag{2.82}$$

p^* bezeichnet hierbei den auf den kritischen Druck bezogenen Siededruck der flüssigen Phase:

$$p^* = \frac{p_S}{p_{\mathrm{krit}}} \tag{2.83}$$

Statt mit dem in Gleichung (2.63) definierten Dampfmassenstromanteil $\dot{x}$ zu rechnen, kann der Zweiphasenmultiplikator nach [20] in guter Näherung auch unter Annahme eines homogenen Gemisches flüssiger und dampfförmiger Phase ermittelt werden. Mit dem Schlupf $S = c''/c' = 1$ gilt dann: $\dot{x} = x$.

3 Stoffwerte

3.1 Wasser

Für die numerische Auswertung werden verschiedene Stoffwerte für flüssiges Wasser und Wasserdampf benötigt. Je nach Formulierung der Bilanzierungsgleichungen sind sie als Funktionen unterschiedlicher Zustandsgrößenpaare zur Verfügung zu stellen.

Am Arbeitsbereich Energietechnik der TU Hamburg-Harburg wurde ein Programmpaket PROST (Properties of Water and Steam) entwickelt, das Stoffwerte von Wasser und Wasserdampf bei Vorgabe verschieden wählbarer Paare an Zustandsgrößen bereitstellt. Die Funktionen basieren auf der spezifischen Helmholtzschen Freien Energie, d. h. Zustandsgrößen werden aus den Werten für v und T berechnet. Für andere Vorgabepaare werden entsprechende mit dem Newton-Verfahren iterativ auszuwertende Umkehrfunktionen angeboten. Auch für die Ermittlung von Stoffwerten an den Phasenübergängen $x = 0$ und $x = 1$ stehen, je nachdem ob die Temperatur oder der Druck bekannt sind, die Funktionen `sat_t` und `sat_p` zur Verfügung. Zusätzliche Funktionen ermöglichen die Ermittlung der Wärmeleitfähigkeit λ und der dynamischen Viskosität η von Wasser/Dampf.

Die PROST-Funktionen berechnen die Stoffwerte kontinuierlich, denkbar wäre hier auch zur Einsparung von Rechenzeit die Interpolation von Stoffwerttabellen diskreter Zustandspunkte. Hier sind aber aufgrund des diskontinuierlichen Größenverlaufs Schwierigkeiten bezüglich der Rechenstabilität zu erwarten.

Die Stoffwertefunktionen liefern bei gegebenem Zustandsgrößenpaar (bzw. bei den Sättigungsfunktionen für bekannte Zustandsgröße) einen 14-komponentigen Vektor (bzw. je einen Vektor für $x = 0$ und $x = 1$). Auf die in Tabelle 5 aufgeführten Komponenten kann bei Angabe des Index bzw. der mnemonischen Schreibweise zugegriffen werden.

Index	mnemonische Bezeichnung	Zustandsgröße	Einheit
0	IDX_T	Temperatur	K
1	IDX_D	Dichte	kg/m^3
2	IDX_P	Druck	Pa
3	IDX_F	spezif. freie Energie	J/kg
4	IDX_G	spezif. freie Enthalpie	J/kg
5	IDX_S	spezif. Entropie	J/kg K
6	IDX_U	spezif. innere Energie	J/kg
7	IDX_H	spezif. Enthalpie	J/kg
8	IDX_CV	spezif. Wärmekapazität bei konst. Volumen	J/kg K
9	IDX_CP	spezif. Wärmekapazität bei konst. Druck	J/kg K
10	IDX_W	Schallgeschwindigkeit	m/s
11	IDX_DPT	$(\partial p/\partial T)_\varrho$	Pa/K
12	IDX_DPD	$(\partial p/\partial \varrho)_T$	$Pa/kg\,cm^3$
13	IDX_ERROR	Fehlerflag; 0.0 wenn erfolgreich, sonst 1.0	—

Tabelle 5: Komponenten des Stoffwertevektors für flüssiges Wasser und Wasserdampf

Die Funktionen liefern im Einphasengebiet einen Wert für die spezifische Wärmekapazität, im Naßdampfbereich wird der Wert $c_p = 0,0$ gesetzt. Dies kann als ein Abfragekriterium für die

Lage des Punktes im Zustandsgebiet genutzt werden.

Bei der hier verwendeten Formulierung der Massen- und Energiebilanz sind die Stoffwerte für eine Vorgabe von Enthalpie und Dichte mit der Stoffwertfunktion `whd` zu berechnen. Sie kann sowohl im Ein- als auch im Zweiphasengebiet aufgerufen werden. Die Funktion `whd` kann in SMILE nach Einbinden der Headerdatei „`steam.h`" z. B. zur Berechnung der Temperatur auf folgende Weise eingesetzt werden:

```
@eq cont T[j] (h[j], rho[j])
    {
     double prop[14];
     double T_guess;

     T_guess = T[j][|last|];

        whd (h[j], rho[j], T_guess, 1.0e-12, prop);

        return prop[IDX_T];
    }
```

Es bietet sich an, für den Temperaturschätzwert (in diesem Beispiel `T_guess`), der für den Stoffwerte-Iterationsstart vorzugeben ist, den berechneten Wert für T des letzten Zeitschritts zu übernehmen. Der Vorgabewert für die einzuhaltende Rechengenauigkeit (hier mit `1.0e-12` fest eingetragen) sollte aus Gründen der numerischen Stabilität nicht zu groß gewählt werden. Bei der SMILE-Kompilation ist die Stoffwertebibliothek „`libsteam.a`" einzubinden.

Für die Weiterentwicklung des Stoffwertpakets PROST könnten noch folgende Punkte angedacht werden:

- In der bisherigen Form wird aus einem Zustandsgrößenpaar jeweils der komplette Vektor berechnet. Da jedoch in SMILE je Funktionsaufruf nur einer Zustandsgröße ein Wert zugewiesen werden kann, werden zwangsläufig auf Kosten eines höheren Rechenzeitbedarfs ungewünschte Zustandsgrößen mitberechnet. Denkbar wäre hier das Zurverfügungstellen von Teilfunktionen zur Ermittlung konkreter Zustandsgrößen.

- Die Stoffwertfunktionen unterscheiden intern zwischen den verschiedenen Wasserzustandsgebieten, so daß z. B. sowohl für die Wasserbilanzierung des Verdampfer- als auch des Überhitzerrohres die Funktion `whd` eingesetzt werden kann. Dies kann jedoch zu numerischen Problemen am Phasenübergang führen, da die Verläufe der Zustandsgrößen einen Knick aufweisen. Wünschenswert wäre vor allem für den Übergang an der Siedelinie, den Funktionsverlauf in einem Bereich auf das jeweils andere Gebiet zu extrapolieren und somit Unstetigkeiten in der ersten Ableitung zu vermeiden. Dem Nutzer ist es jedoch nicht möglich zu entscheiden, in welchem Bereich der betrachtete Punkt liegt, ohne daß ein Stoffwertfunktionsaufruf ausgewertet wurde. Eine Aufteilung in Stoffwertteilfunktionen für die jeweiligen Bereiche würde die angesprochene Extrapolation ermöglichen (siehe hierzu auch Kapitel 5.2).

Die SMILE-Implementation der Stoffwertefunktionsklassen für Wasserdampf SWFUebhDampfPUndTAusHUndRho bzw. SWFUebhDampfPUndTAusHUndRhoIdGas und SWFNassdampfPUndTAusHUndRho ist in Anhang B.6.1 ab Seite 104 zu finden. Abbildung 23 auf Seite 105 zeigt die Klassenstruktur. Der Tabelle 15 auf Seite 105 können die jeweiligen `@export`-Variablen entnommen werden.

3.2 Rauchgas

Die Stoffwerte von Luft und Rauchgas können als reine Funktion der Temperatur angesehen werden. Das Rauchgas wird in den in Dampferzeugern üblichen Druck- und Temperaturbereichen in guter Näherung als ein Gemisch idealer Gase betrachtet.

Für SMILE steht mit dem von Hubertus Tummescheit geschriebenen Modell Ideal_Gas_Caloric_2 ([58]) eine Klasse zur Verfügung, die in Abhängigkeit von der Rauchgaszusammensetzung die Gaskonstante des Rauchgases und daraus die Dichte und die spezifische Wärmekapazität berechnet. Es werden die möglichen Einzelkomponenten Ar, Ne, N_2, O_2, CO, CO_2, H_2O, SO_2 und Luft berücksichtigt.

Damit Stoffwerte bei genaueren Rechnungen in entsprechenden SMILE-Modellen ermittelt und in den Bilanzierungsgleichungen verwendet werden können, sind sie in den Bilanzklassen als `@export`-Größen implementiert. So können sie über `@connects` aus den Berechnungsklassen übernommen werden. Zunächst werden sie aber zur Validierung der Dampferzeugerkomponenten als konstant und gegeben angesehen.

4 Modellierung eines Dampferzeugers

4.1 Einzelkomponenten

Im folgenden wird die Modellierung einzelner Dampferzeugerbestandteile beschrieben. Bevor komplexere Systeme wie z. B. der Feuerraum oder das wasser- bzw. dampfdurchströmte Rohr simuliert werden, sollen nach einer grundlegenden Betrachtung über die Richtung des Informationsflusses zunächst zwei einfache Grundbestandteile beschrieben werden, die vor allem für die Kombination der verschiedenen Einzelkomponenten erforderlich sind: die Drossel und der Massespeicher.

4.1.1 Kombination der Einzelkomponenten

Die Gleichungen der Dampferzeugereinzelkomponenten sind so formuliert, daß das Bauteil eindeutig einem von zwei Typen zugeordnet werden kann:

- Typ „*Speicher*“: Alle die Bilanzgrenzen überschreitenden Massenströme sind vorzugeben. Aus diesen Größen wird der Druck berechnet. Zu diesem Typ zählen z. B. die Komponenten Dampferzeugerrohr oder Dampftrommel. Genauso kann ein Modell, das Wärme (und nicht Masse) speichert, dem Typ „Speicher“ zugeordnet werden. Bei Vorgabe von Wärmeleistungen können dann Temperaturen bestimmt werden. Diesem Typ sind beispielsweise die Modelle DiscreteHeatStore oder Waermeleitung zugehörig.
- Typ „*Drossel*“: Die Massenströme an den Schnittstellen werden aus den vorgegebenen Drücken ermittelt. Vom Typ „Drossel“ sind z. B. die Komponenten Pumpe oder Drossel. Auch vom Typ „Drossel“ gibt es eine den Wärmestrom betreffende Variante: Beim Wärmeübergang oder der Wärmestrahlung sind z. B. die Temperaturen vorzugeben, damit die Wärmeleistungen berechnet werden können — man spricht in diesem Zusammenhang auch von „Wärmewiderstand“.

Während die Richtung des Informationsflusses von Drücken und Massenströmen vom Bauteiltyp abhängt, ist die Richtung des Informationsflusses der Energien in jedem Fall an die Strömungsrichtung gekoppelt. Die spezifischen Enthalpien sind also am Strömungseintritt vorzugeben, und berechnete Enthalpien werden am Strömungsaustritt an die folgende Komponente weitergereicht. Schwierigkeiten bereitet dieses Prinzip nur, wenn die Strömungsrichtung umkehrt: Da sich somit auch die Informationsrichtung des Energieflusses umdreht, werden vorher vorzugebende Variablen jetzt zu berechneten Größen.

Um Schwierigkeiten mit den `@connects` zu vermeiden, wird bei der Kombination der Dampferzeugereinzelkomponenten das „Prinzip der bunten Reihe“ eingehalten: Dampferzeugerbestandteile sind in abwechselnder Reihenfolge vom Typ „Speicher“ und vom Typ „Drossel“ aneinanderzufügen. Ein Dampferzeugerrohr kann z. B. direkt mit der Dampftrommel verknüpft werden, die Verbindung zu einem weiteren Dampferzeugerrohr kann aber nur über ein Bindeglied des Typs „Drossel“ hergestellt werden. Hier findet z. B. das einfache Modell Drossel Verwendung. Für die Verknüpfung zweier Komponenten des Typs „Drossel“ kann der in Kapitel 4.1.3 vorgeschlagene Massespeicher eingesetzt werden.

Das „Prinzip der bunten Reihe“ soll auch auf das gesamte Kraftwerk angewendet werden, z. B. Dampferzeuger vom Typ „Speicher“, Dampfturbine vom Typ „Drossel“.

4.1.2 Drossel

Für die isenthalpe Drosselung gilt die Energiegleichung

$$\dot{H}_1 = \dot{H}_2 \tag{4.1}$$

Vernachlässigt man die Massespeicherfähigkeit, so gilt zu jedem Zeitpunkt $\dot{m}_1 = \dot{m}_2$. Damit kann die Energiebilanz auch in spezifischen Größen geschrieben werden:

$$h_1 = h_2 \tag{4.2}$$

Mit

$$\dot{m} = \varrho \cdot A \cdot c \tag{4.3}$$

und

$$\Delta p = \zeta \cdot \varrho \cdot \frac{\mathrm{c}^2}{2} \tag{4.4}$$

bzw. umgeformt

$$c = \sqrt{\frac{2 \cdot \Delta p}{\zeta \cdot \varrho}} \tag{4.5}$$

ergibt sich die gesuchte Beziehung $\dot{m} = f(\Delta p)$ zu

$$\dot{m} = A \cdot \sqrt{\frac{2 \cdot \varrho \cdot \Delta p}{\zeta}} \tag{4.6}$$

bzw. einfacher mit $k = A \cdot \sqrt{\frac{2 \cdot \varrho}{\zeta}}$:

$$\dot{m} = k \cdot \sqrt{\Delta p} \tag{4.7}$$

Auch diese Funktion soll, wie bereits in Kapitel 2.6 beschrieben, in Teilfunktionen untergliedert werden, damit der Verlauf von $\dot{m} = f(\Delta p)$ überall stetig und stetig differenzierbar ist. Man erhält:

$$\dot{m} = \dot{m}_1 = \dot{m}_2 = \begin{cases} -k \cdot \sqrt{-\Delta p} & \Delta p < -1,0 \\ -\frac{k}{4} \cdot ((\Delta p)^3 + 5 \cdot \Delta p) \qquad \text{für} & -1,0 \leq \Delta p \leq 1,0 \\ k \cdot \sqrt{\Delta p} & 1,0 < \Delta p \end{cases} \tag{4.8}$$

Da in der Drossel maximal Schallgeschwindigkeit erreicht werden kann, muß noch eine durch das kritische Druckverhältnis Π_{krit}[4] vorgegebene Massenstrombegrenzung berücksichtigt werden. Weil in Dampferzeugern jedoch das vorhandene immer größer ist als das kritische Druckverhältnis $\Pi = \frac{p_{\mathrm{aus}}}{p_{\mathrm{ein}}} > \Pi_{\mathrm{krit}}$, ist hier auf die Beachtung der Limitierung verzichtet worden.

[4]$\Pi_{\mathrm{krit}} = \left.\frac{p_{\mathrm{aus}}}{p_{\mathrm{ein}}}\right|_{\mathrm{krit}} = f(\kappa)$. Für alle $\Pi < \Pi_{\mathrm{krit}}$ nimmt $\dot{m}$ einen konstanten Maximalwert an.

Die SMILE-Implementation der Drossel ist in Anhang B.8.2 ab Seite 126 zu finden. Der Tabelle 16 auf Seite 111 können die `@export`-Variablen entnommen werden.

4.1.3 Massespeicher

Der Massespeicher stellt das Gegenstück zur im vorigen Kapitel beschriebenen Drossel dar: Während sich vorher aufgrund einer vorgegebenen Druckdifferenz ein Massenstrom einstellte, sind hier alle ein- und austretenden Massenströme Ursache für den sich einstellenden Druck.

In Abbildung 7 ist ein Massespeicher mit einer Massenstromzu- und einer -abführung dargestellt.

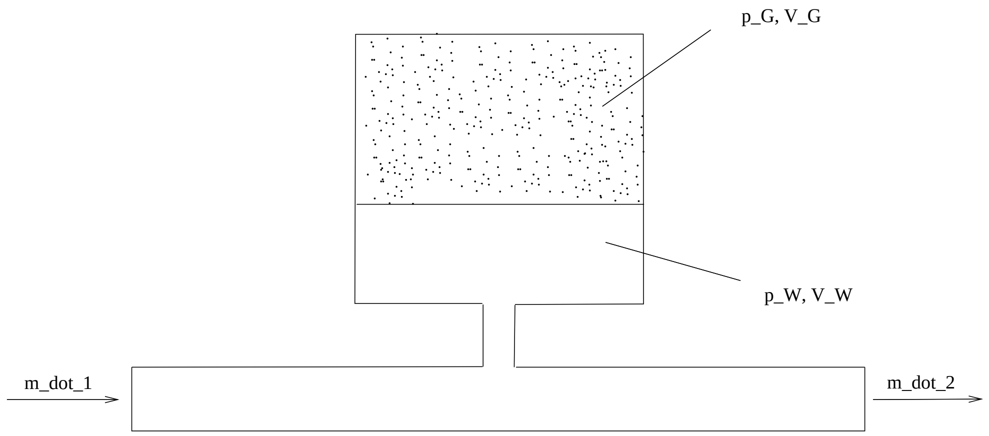

Abbildung 7: Massespeicher

Der Massenstrom 1 wird positiv gezählt, wenn er eintritt, Massenstrom 2, wenn er austritt. Im Gesamtspeichervolumen V_0 wird das Gasvolumen V_G vom vom Wasser eingenommenen Volumen V_W getrennt. Für isotherme Zustandsänderungen des ideal angenommenen Gases gilt: $p \cdot V = \text{const} = k$. Der gemeinsame Druck $p = p_G = p_W$ kann aus der Konstante k und dem vom Gas beanspruchten Volumen bestimmt werden. Mit dem Wasservolumen $V_W = \frac{m}{\varrho}$ gilt:

$$p = \frac{k}{V_G} = \frac{k}{V_0 - V_W} = \frac{k}{V_0 - \frac{m}{\varrho}} \tag{4.9}$$

Die Änderung der Wassermasse im Speicher ergibt sich aus der Summe der ein- und austretenden Massenströme:

$$\frac{\partial m}{\partial t} = \dot{m}_1 - \dot{m}_2 \tag{4.10}$$

Die SMILE-Implementation des Massespeichers ist in Anhang B.5.1 ab Seite 103 zu finden.

4.1.4 Feuerraum

Ein einfaches Brennraummodell ist das nulldimensionale Berechnungsmodell nach Konakow [39]. Es geht von folgenden Annahmen aus:

- Der Verbrennungsprozeß läuft unendlich schnell ab.
- Die Rauchgastemperatur ist für einen Lastfall im gesamten Feuerraum konstant.

- Wärme wird nur durch Strahlung übertragen.

Ausgangspunkt für das Konakow-Modell ist die angenäherte Feuerraumbilanzierung

$$\dot{m}_{\mathrm{Rg}} \cdot c_{\mathrm{p,Rg}} \cdot (T_{\mathrm{ad}} - T_{\mathrm{FA}}) = \varepsilon \cdot C_S \cdot A \cdot (T_{\mathrm{g}}^4 - T_{\mathrm{W}}^4) \tag{4.11}$$

Das Modell berücksichtigt nicht die exakte Verteilung der Wärmestromdichte, sondern die bedeutendere integrale Wärmezufuhr im Feuerraum.

Die Strahlungstemperatur der Rauchgasstrahlung T_{g} kann über das in guter Näherung als konstant angenommene Temperaturverhältnis a aus der Temperatur der Verbrennungsgase am Feuerraumaustritt T_{FA} bestimmt werden:

$$T_{\mathrm{g}} = a \cdot T_{\mathrm{FA}} \tag{4.12}$$

Der „konventionelle Schwärzefaktor“ z berechnet sich nach

$$z = a \cdot \sqrt[4]{\varepsilon} \tag{4.13}$$

bzw.

$$a = \frac{z}{\sqrt[4]{\varepsilon}} \tag{4.14}$$

Er ist für unterschiedliche Feuerungsarten nur wenig verschieden und streut für sie jeweils nur schwach. Nach [39] kann für verschiedene Feuerungsarten mit dem in Tabelle 6 angegebenen Schwärzefaktor gerechnet werden. Angegeben ist hierbei zusätzlich jeweils die mittlere relative Streuung v.

Aus der Energiebilanz für die Reaktionszone

$$\begin{aligned} \dot{m}_{\mathrm{Br}} \cdot (H_u(T_0) + c_{\mathrm{p,Br}} \cdot (T_{\mathrm{Br}} - T_0)) + \dot{m}_{\mathrm{L}} \cdot c_{\mathrm{p,L}} \cdot (T_{\mathrm{L}} - T_0) = \\ \dot{m}_{\mathrm{Rg}} \cdot c_{\mathrm{p,Rg}} \cdot (T_{\mathrm{ad}} - T_0) + \dot{m}_A \cdot c_{\mathrm{p,A}} \cdot (T_{\mathrm{ad}} - T_0) + \dot{Q}_{\mathrm{unv}} \end{aligned} \tag{4.15}$$

kann die adiabate Verbrennungstemperatur T_{ad} bestimmt werden. Der Austrittsenthalpiestrom der im Rauchgas enthaltenen Asche $\dot{m}_A \cdot c_{\mathrm{p,A}} \cdot (T_{\mathrm{ad}} - T_0)$ wird vernachlässigt, und die Verbrennung wird als vollständig angesehen ($\dot{Q}_{\mathrm{unv}} = 0$). Mit $h_{\mathrm{Br}} = c_{\mathrm{p,Br}} \cdot (T_{\mathrm{Br}} - T_0)$, $T_0 = 273,15$ K und

Feuerungsart	z [—]	v [%]
Schwerölfeuerungen	0,970	4,7
Rostfeuerungen	0,955	5,9
Gasfeuerungen	0,920	6,5
Staubfeuerungen für		
Anthrazit und Magerkohle	0,900	6,2
Steinkohle	0,933	6,5
Braunkohle, Torf und Brennschiefer	0,872	7,4

Tabelle 6: Schwärzefaktor z und Variationskoeffizient v für verschiedene Feuerungsarten

$H_u = H_u(T_0 = 273,15\mathrm{K})$ ergibt sich die adiabate Verbrennungstemperatur zu

$$T_{\mathrm{ad}} = \frac{\dot{m}_{\mathrm{Br}} \cdot (H_{\mathrm{u}} + h_{\mathrm{Br}}) + \dot{m}_{\mathrm{L}} \cdot c_{\mathrm{p,L}} \cdot (T_{\mathrm{L}} - 273,15)}{\dot{m}_{\mathrm{Rg}} \cdot c_{\mathrm{p,Rg}}} + 273,15 \tag{4.16}$$

Einen Ähnlichkeitsparameter der Feuerung stellt die dimensionslose Konakow-Zahl dar:

$$\mathrm{Ko} = \frac{\dot{m}_{\mathrm{Rg}} \cdot c_{\mathrm{p,Rg}}}{z^4 \cdot C_S \cdot A \cdot T_{\mathrm{ad}}^3} \tag{4.17}$$

Geht man davon aus, daß die Wandtemperatur gegenüber der Strahlungstemperatur vernachlässigbar klein ist, $(T_{\mathrm{W}}/T_{\mathrm{g}})^4 \ll 1$, so kann die Feuerraumaustrittstemperatur T_{FA} damit iterativ aus der Gleichung

$$T_{\mathrm{FA}} = T_{\mathrm{ad}} \cdot \left(1 - \frac{1}{\mathrm{Ko}} \cdot \left(\frac{T_{\mathrm{FA}}}{T_{\mathrm{ad}}}\right)^4\right) \tag{4.18}$$

ermittelt werden.

Die SMILE-Implementation der Klasse WueFeuerraum steht in Anhang B.3.2 ab Seite 93. Abbildung 21 auf Seite 84 zeigt die Integration in die Struktur der Wärmeübertragungsklassen. Der Tabelle 11 auf Seite 85 können die `@export`-Variablen entnommen werden.

4.1.5 Wärmetauscher

Die Simulation von Wärmetauschern beruht auf dem Einrohrmodell (siehe hierzu z. B. [2], S. 51). Es reduziert das Rohrbündel auf ein einzelnes Rohr (bzw. auf ein Rohrpaket gleichmäßig gleichsinnig parallel durchströmter Rohre), bei dem das wärmeabgebende und das wärmeaufnehmende Medium entweder im Gleich- oder im Gegenstrom geführt werden. Geht man bei beiden Fluiden von einer Indizierung der Diskretisierungselemente in jeweils positiver Strömungsrichtung aus, so kann eine gleichsinnige Strömung im SMILE-Programmcode durch einfache `@connects`, z. B.

```
wand.T1[i] = wue_aussen.t_wall[i];
```

und eine gegensinnige Strömung durch ensprechend entgegengerichtete `@connects` auf einer der beiden Wärmeübertragungsseiten, z. B.

```
wand.T1[0] = wue_aussen.t_wall[ND-1];
wand.T1[1] = wue_aussen.t_wall[ND-2];
      ⋮
wand.T1[ND-1] = wue_aussen.t_wall[0];
```

simuliert werden. Eine zum Gleichstromwärmeaustauscher analoge Formulierung zur Erfassung aller Diskretisierungselemente durch eine einzige `@connect`-Verbindung der Form

```
wand.T1[i] = wue_aussen.t_wall[(ND-1)-i];
```

ist in SMILE leider (noch) nicht möglich.

Die wärmespeichernde Masse und die Oberfläche der Rohre stimmen im Modell mit denen des nachzubildenden Rohrbündels überein. Vorrichtungen zur Vergrößerung der Rohroberfläche, wie z. B. Rohrberippung, können in der Modellierung durch einen Flächenvergrößerungsfaktor berücksichtigt werden. Durch die vereinfachende Betrachtungsweise des Einrohrmodells werden weder die Querschnittsfläche der Rohre noch die rohrinnenseitigen Druckverluste verfälscht. Die Queranströmung der Rohre sowie die fluchtende bzw. versetzte Anordnung der einzelnen Rohre werden durch die unterschiedlichen Wärmeübergangsbeziehungen berücksichtigt.

4.1.6 Dampferzeugerrohr

In diesem Kapitel soll ein Dampferzeugerrohr (bzw. gemäß den vorangegangenen Betrachtungen ein Dampferzeugerrohrbündel) als Spezialfall eines Wärmetauschers betrachtet werden.

Um die Einzelkomponenten des Dampferzeugers austauschbar zu machen, erben die Modelle Ecorohr, Ueberhitzerrohr und Verdampferrohr aus einer gemeinsamen Schnittstellenklasse RgWasserWAT. Der allgemeine Rauchgas-Wasser-Wärmeaustauscher besteht aus den Komponenten Bilanzierung der Rauchgasseite, Wärmeübertragung/Wärmespeicherung und Bilanzierung der Wasserseite (siehe Abbildung 8). Die Pfeile beschreiben in dieser Darstellung die Richtung des Informationsflusses. Der Pfeil `m_dot_wasser_out` zeigt beispielsweise in den Rauchgas-Wasser-Wärmeaustauscher hinein. Obwohl der Massenstrom bezüglich der Strömungsrichtung austritt, stellt er eine Vorgabegröße dar, aus der die den Informationsfluß betreffenden Austrittsgrößen zu ermitteln sind.

Während sich die Modelle für die Bilanzierung der Rauchgasseite und die Wärmeübertragung für die unterschiedlichen Rohrtypen nicht wesentlich unterscheiden, ist die Bilanzierung der Wasserseite für Ecorohr, Überhitzerrohr und Verdampferrohr getrennt zu formulieren. Auch hier stellt eine Klasse Wasserbilanz_cuts zunächst gemeinsame Schnittstellen zur Verfügung.

In Abbildung 9 ist die Struktur der Wasserbilanzierung eines Verdampfer- bzw. Überhitzerrohres dargestellt. Die Bilanzierungs- und Stoffwertemodelle können sich unterscheiden, gemeinsame Schnittstellen gewährleisten hier aber eine einfache Austauschbarkeit der Teilmodelle.

Das Ecorohr ist bezüglich der Struktur der Wasserbilanzierung gesondert zu betrachten, da es aufgrund der Inkompressibilität flüssigen Wassers eine von Verdampfer- und Überhitzerrohr deutlich abweichende Gleichungsformulierung erfordert. Dessen Struktur ist in Abbildung 10 dargestellt.

Die Wasserbilanz am Dampferzeugerrohr soll sowohl beim Verdampfer- und Überhitzer- als auch am Ecorohr durch ein Modell vom (in Kapitel 4.1.1 erläuterten) Typ „Speicher“ beschrieben werden. Das heißt, daß am Ein- und Austritt des Rohres die Massenströme vorzugeben sind. Hieraus werden die sich einstellenden Drücke berechnet. Der Informationsfluß der Energie ist an die Strömungsrichtung gebunden: Die Eintrittsenthalpie muß bekannt sein, hieraus wird die Enthalpie am Rohraustritt berechnet.

4.1.6.1 Verdampferrohr Ein Verdampferrohr soll als Spezialfall des allgemeinen Dampferzeugerrohres simuliert werden. Es setzt sich zusammen aus den in Abbildung 9 dargestellten Komponenten

- Wasserbilanz,
- Wärmeübergang auf der Arbeitsmittelseite,
- Wärmeleitung bzw. -speicherung,
- Wärmeübergang auf der Rauchgasseite und

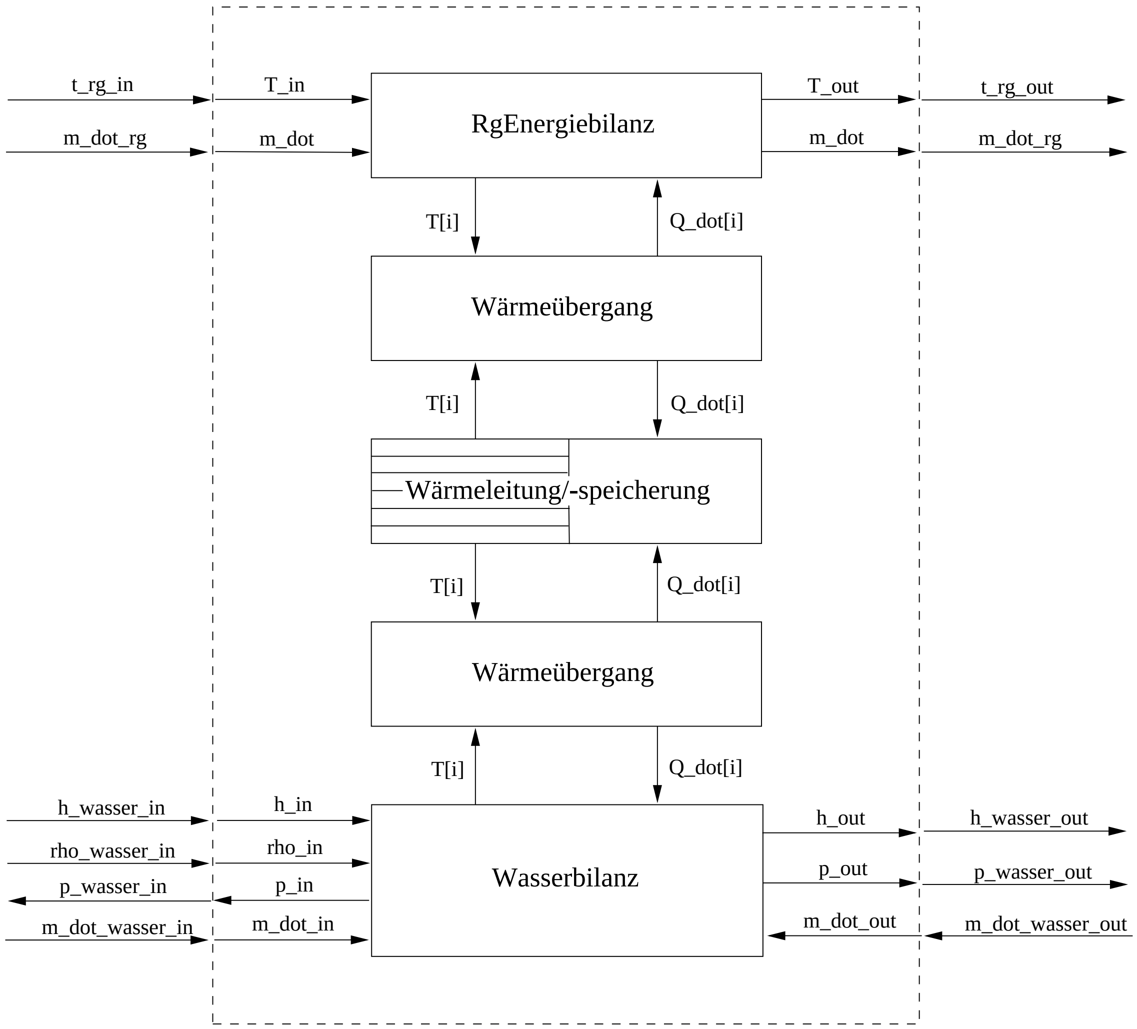

Abbildung 8: Struktur des Rauchgas-Wasser-Wärmeaustauschers

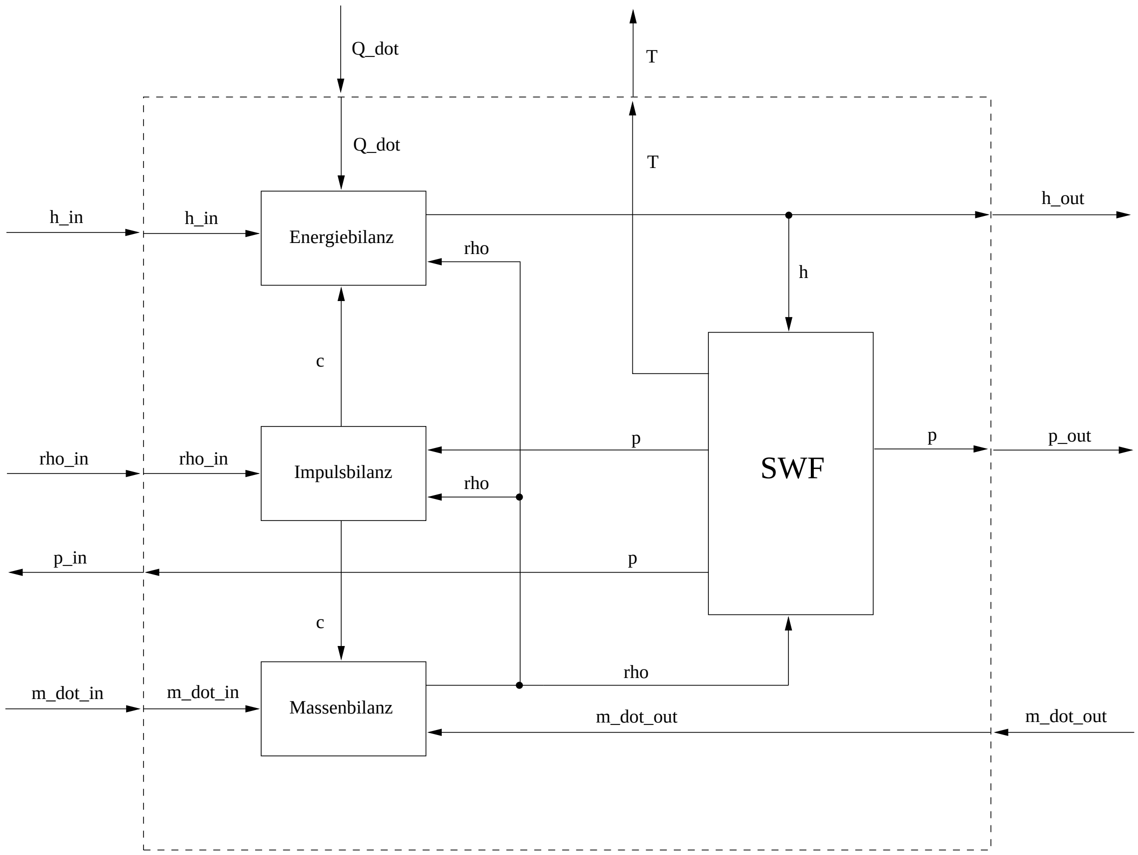

Abbildung 9: Struktur der Wasserbilanzierung von Verdampfer- und Überhitzerrohr mit Bilanzgleichungen und Stoffwertefunktion

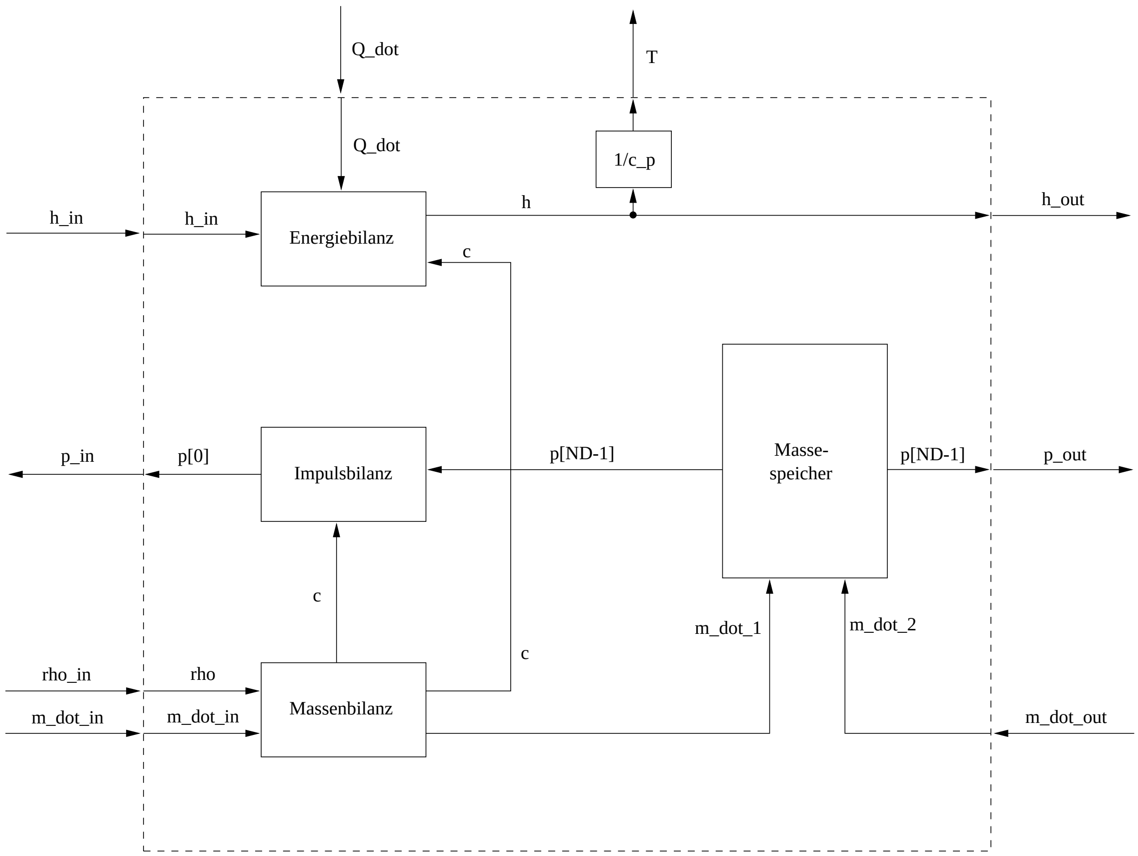

Abbildung 10: Struktur der Wasserbilanzierung vom Ecorohr mit Bilanzgleichungen und Massespeicher

- Energiebilanzierung des Rauchgases.

Für die Beschreibung der Dampfseite werden in der Klasse WasserbilanzVerdampferrohr die Massen-, Energie- und Impulsbilanz in ihrer jeweils instationären Formulierung vereint. Die Energiegleichung ist in der Enthalpieformulierung zu verwenden, da Naßdampf nicht als ideales Gas betrachtet werden kann. Da Dichte und Geschwindigkeit in Strömungsrichtung deutlich ab- bzw. zunehmen können, ist mit den Modellen verteilter Massen (DM) zu rechnen. Auf der Dampfseite wird mit der Wärmeübergangsklasse WueVerdampfungRohr gerechnet, rauchgasseitig je nach Temperaturniveau mit WueStrahlungUndKonvektionRohr bzw. WueKonvektionRohr. Die jeweiligen Stoffwerte werden aus den Zustandsgrößen Enthalpie und Dichte unter Aufruf der Stoffwertefunktion (SWF) `whd` ermittelt. Zu Kontrollzwecken können hier zusätzlich die Sättigungswerte für $x = 0$ bzw. $x = 1$ aus Auswertungen der Funktion `sat_t` oder `sat_p` zu gegebener Temperatur bzw. Druck bestimmt werden. Deren Berechnung ist für die Gewährleistung der Funktion jedoch nicht erforderlich.

Da auf gemeinsame Schnittstellen geachtet wurde, kann durch einfaches Aus- bzw. Entkommentieren im Programmcode mit unterschiedlichen Komponentenmodellen die Modellierungstiefe dem betrachteten Gesamtsystem angepaßt werden. So kann beispielsweise, um Rechenaufwand für die α-Bestimmung zu vermeiden, statt der Klasse WueVerdampfungRohr das einfachere Modell WueKonvektion unter Vorgabe eines konstanten (und bei Verdampfung entsprechend großen) Wärmeübergangskoeffizienten α simuliert werden.

Die SMILE-Implementation der Verdampferrohrklassen steht in Anhang B.7.3 ab Seite 118. Abbildung 25 auf Seite 113 zeigt die Klassenstruktur. Der Tabelle 16 auf Seite 111 können die jeweiligen `@export`-Variablen entnommen werden.

4.1.6.2 Überhitzerrohr Das Überhitzerrohr ist dem Verdampferrohr sehr ähnlich aufgebaut: Die Bilanzierungsgleichungen beider Medien sind dieselben, und auch die benötigten Stoffwerte Druck und Temperatur werden aus den Zustandsgrößen Enthalpie und Dichte mit der gleichen Funktion `whd` bestimmt. Qualitative Unterschiede ergeben sich nur in den dampfseitigen Wärmeübergangsbeziehungen.

Die vereinfachende Betrachtungsweise von überhitztem Dampf als idealem Gas gestattet eine zusätzliche Option in der Ermittlung der Stoffwerte. Die Güte der Annahme steigt mit wachsender Dampftemperatur. Die Temperatur wird hier nach

$$T = \frac{h - h''}{c_p} + T_S \tag{4.19}$$

mit $c_p = 2200\ \frac{\mathrm{J}}{\mathrm{kgK}}$ und der Druck aus der Gleichung

$$p = R \cdot T \cdot \varrho \tag{4.20}$$

mit $R = 462{,}0\ \frac{\mathrm{J}}{\mathrm{kgK}}$ bestimmt.

Ebenso wie das Überhitzerrohr kann auch der Zwischenüberhitzer simuliert werden; es bestehen keine qualitativen, sonder nur quantitative Unterschiede.

Die SMILE-Implementation der Überhitzerrohrklassen ist in Anhang B.7.4 ab Seite 121 zu finden. Abbildung 26 auf Seite 114 zeigt die Klassenstruktur. Der Tabelle 16 auf Seite 111 können die jeweiligen `@export`-Variablen entnommen werden.

4.1.6.3 Ecorohr Die Formulierung der Wasserbilanzgleichungen unterscheidet sich beim Ecorohr von der des Verdampfer- und Überhitzerrohres aufgrund der Inkompressibilität von flüssigem Wasser. Während die Energiebilanz ähnlich wie beim Verdampfer- und Überhitzerrohr formuliert werden kann, reduziert sich die Impulsbilanz auf eine Bestimmung des Druckverlustes. Er wird sinnvollerweise nicht für jedes diskrete Element, sondern für das gesamte Rohr ermittelt. Die Ermittlung der zeitlichen Dichteänderung in der Massenbilanz ist nicht mehr zweckmäßig — es kann mit einem konstanten Wert gerechnet werden. Da beim allgemeinen Dampferzeugerrohr sowohl Ein- als auch Austrittsmassenstrom vorgegeben werden sollen, die Kenntnis der Wasserinkompressibilität aber gleich große Massenströme verlangt, ist an das Ecorohr ein Speicher angehängt. Er läßt auf Kosten eines sich verändernden Druckes zu, daß Ein- und Austrittsmassenstrom unterschiedlich groß sein dürfen. Auch die Berechnung der Stoffwerte vereinfacht sich: Die Temperatur kann in erster Näherung als reine Funktion der Enthalpie angesehen werden. Die Temperatur in K ergibt sich bei konstanter Wärmekapazität $c_\mathrm{W} = 4200 \ \frac{\mathrm{J}}{\mathrm{kgK}}$ aus der spezifischen Enthalpie h zu

$$T_i = \frac{h_i}{c_\mathrm{W}} + 273,15 \ \mathrm{K} \tag{4.21}$$

Die SMILE-Implementation der Ecorohrklassen ist in Anhang B.7.2 ab Seite 115 zu finden. Abbildung 24 auf Seite 112 zeigt die Klassenstruktur. Der Tabelle 16 auf Seite 111 können die jeweiligen `@export`-Variablen entnommen werden.

4.1.7 Einspritzkühler

Vernachlässigt man die Massenspeicherfähigkeit des Einspritzkühlers, so beschreiben die Massenbilanz

$$\dot{m}_\mathrm{in} = \dot{m}_\mathrm{out} - \dot{m}_\mathrm{w} \tag{4.22}$$

und die Energiebilanz

$$h_\mathrm{out} = \frac{\dot{m}_\mathrm{in} \cdot h_\mathrm{in} + \dot{m}_\mathrm{w} \cdot h_\mathrm{w}}{\dot{m}_\mathrm{out}} \tag{4.23}$$

dessen thermisches Verhalten. Sollen später jedoch im ganzen Kraftwerksmodell alle `@connects`,

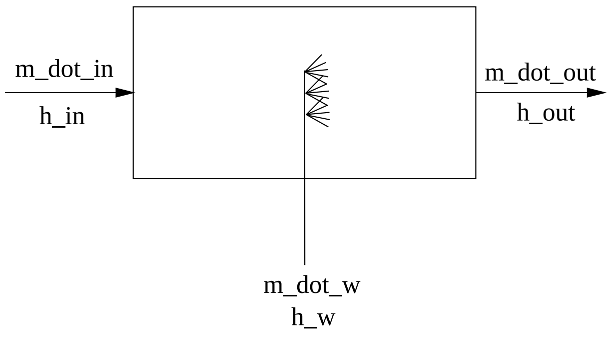

Abbildung 11: Einspritzkühler

also z. B. auch zwischen Speisewasserbehälter und Einspritzkühler, hergestellt werden, so ist es erforderlich, eine weitere unbekannte Größe einzuführen, damit die Anzahl von Gleichungen und Variablen übereinstimmen: die Speichermasse m des Einspritzkühlers. Damit lauten die instationäre Massenbilanz

$$\frac{\partial m}{\partial t} = \dot{m}_{\mathrm{in}} + \dot{m}_{\mathrm{w}} - \dot{m}_{\mathrm{out}} \tag{4.24}$$

und die instationäre Energiebilanz

$$\frac{\partial h}{\partial t} = \frac{\dot{m}_{\mathrm{in}} \cdot h_{\mathrm{in}} + \dot{m}_{\mathrm{w}} \cdot h_{\mathrm{w}} - \dot{m}_{\mathrm{out}} \cdot h_{\mathrm{out}} + \dot{Q}}{m} \tag{4.25}$$

Die SMILE-Implementation der Klasse Einspritzkuehler ist in Anhang B.8.1 ab Seite 125 zu finden. Der Tabelle 16 auf Seite 111 können die jeweiligen `@export`-Variablen entnommen werden.

4.2 Kompletter Dampferzeuger

4.2.1 Dampferzeuger in GuD-Prozessen

Dampferzeuger in GuD-Anlagen stellen das Bindeglied zwischen dem Gas- und dem Dampfprozeß dar. Nach der Art der Feuerung werden drei Konfigurationen unterschieden:

- Abhitzekessel ohne Zusatzfeuerung
- Abhitzekessel mit geringer Zusatzfeuerung
- Dampferzeuger mit maximaler Zusatzfeuerung

Eine typische Anlagenkonfiguration eines zusatzbefeuerten Abhitzekessels ist in Abbildung 12 dargestellt.

Die verschieden möglichen Konfigurationen sind auch in der SMILE-Modellierung unterschiedlich zu berücksichtigen. Bei der Modellierung der gering bzw. maximal zusatzgefeuerten Abhitzekessel ist ein entsprechender Feuerraum einzubeziehen. Wird ein Abhitzekessel unter Ausnutzung des größten Teils des Restsauerstoffgehalts der Gasturbinenabgase maximal zusatzbefeuert, so läßt sich ein konventioneller Dampfprozeß anschließen. Ein besonderes Merkmal des Dampferzeugers ist die evtl. vorhandene Zwischenüberhitzerberohrung. Da sich in dieser Anlagenkonfiguration auch Kohle als Brennstoff einsetzen läßt, ist hier eine entsprechende Feuerraumgestaltung bzw. Brennstoffaufbereitung zu modellieren.

4.2.2 Berücksichtigung des Umlaufsystems

Bei Dampferzeugern können folgende Umlaufsysteme unterschieden werden:

- Umlaufkessel
  - Naturumlauf
  - Zwangumlauf
- Zwangdurchlaufkessel
  - Bensonkessel
  - Sulzerkessel

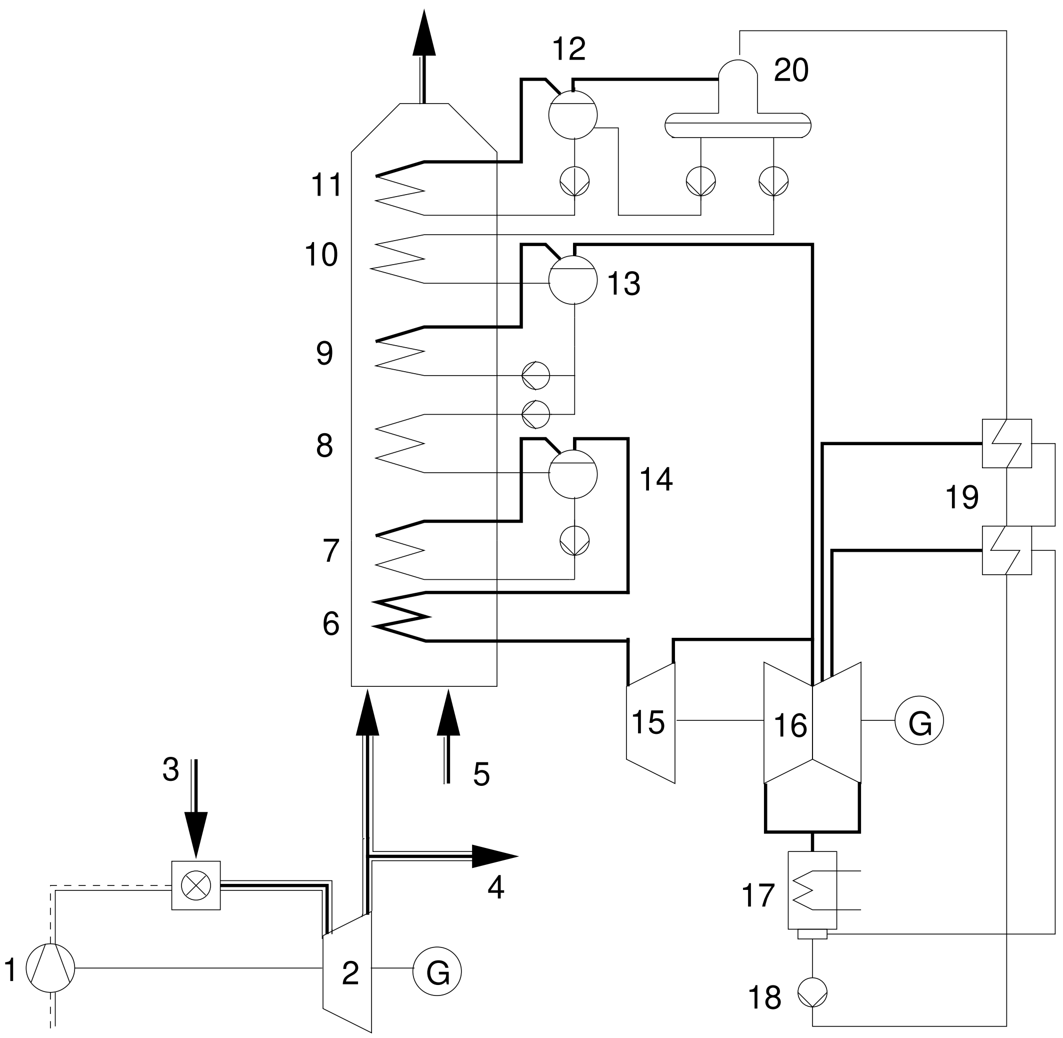

Abbildung 12: Schaltschema eines Abhitzekessels mit geringer Zusatzfeuerung in Zweidruckschaltung (Quelle: [55], S. 7)

1	Verdichter	2	Gasturbine	3	Brennkammer
4	Rauchgasbypass	5	Zusatzfeuerung	6	HD-Überhitzer
7	HD-Verdampfer	8	HD-Economiser	9	MD-Verdampfer
10	MD-Economiser	11	ND-Verdampfer	12	ND-Kesseltrommel
13	MD-Kesseltrommel	14	HD-Kesseltrommel	15	HD-Dampfturbine
16	MD-Dampfturbine	17	Kondensator	18	Kondensatpumpe
19	Speisewasservorwärmer	20	Speisewasserbehälter/Entgaser		

Je nach Umlaufsystem und Kesselbauart sind für die Modellierung des kompletten Dampferzeugers unterschiedliche Teilkomponenten zu berücksichtigen.

Den *Umlauf*dampferzeugern gemeinsam ist die Dampftrommel. Sie trennt nach erfolgter Teilverdampfung den Sattdampf vom flüssigen Medium. Je nach Antrieb für die Durchströmung und Kühlung der beheizten Verdampferrohre werden Naturumlauf- von Zwangumlaufdampferzeugern unterschieden. Bei den Naturumlaufkesseln ist die örtliche Dichteänderung im verdampfenden Medium Ursache für die Durchströmung. In der Energie- und Impulsbilanzierung der Wasserseite ist der Schwerefeldterm zu berücksichtigen. Bei Zwangumlaufdampferzeugern sorgt eine zusätzlich zu modellierende Umwälzpumpe für den Wasserumlauf im Verdampfungskreis.

Der *Zwangdurchlauf*dampferzeuger stellt das in der Modellierung vermutlich komplizierteste System dar. Da die Grenzen für den Verdampfungsbeginn örtlich nicht in einer Dampftrommel festgelegt sind, können die wärmeübertragenden Rohre nicht mehr aus klar unterteilten Bestandteilen Eco-, Verdampfer- und Überhitzerrohr jeweils konstanter Länge zusammengesetzt werden. Zur Modellierung dieser Dampferzeugerrohre bestehen zwei grundsätzliche Möglichkeiten:

- Das Rohr wird längenkonstant modelliert. Damit kann sich der Punkt des Verdampfungsbeginns bzw. -endes innerhalb des Rohres verschieben. Die Wasserbilanzierung und die Wärmeübergangsbeziehungen sind gemäß des Übergangs vom flüssigen Zustand in das Naßdampfgebiet bzw. in den überhitzten Bereich innerhalb des Modells zu wechseln. Das heißt, daß im Modell eine Abfrage nach der Lage im Zustandsgebiet vorzusehen ist. Je nach Zustand des Wassers ist dann zwischen den anzuwendenden Beziehungen, z. B. für die Berechnung des Wärmeübergangskoeffizienten α, zu schalten. Zur ausreichend genauen Beschreibung ist für diesen Fall eine sehr feine örtliche Unterteilung notwendig.
- Die Sättigungszustände werden, wie bei den in Kapitel 4.1.6 beschriebenen Einzelkomponenten Eco-, Verdampfer- und Überhitzerrohr, jeweils an den Enden der Rohre angenommen. In diesem Fall ist aber ein Rohr variabler Länge zu modellieren, das der jeweiligen Verdampfungszone anzupassen ist. Eine entsprechende Umformulierung der Gleichungen ist z. B. in [57] zu finden.

Die Speisepumpe bewirkt die Durchströmung der kompletten Kesselberohrung. Für den Sulzerkessel ist ein Wasserabscheider zu modellieren, beim Bensonkessel wird das Verdampfungsende in einen Restverdampfer verlagert.

4.2.3 Notwendigkeit einer Dampferzeuger-Oberklasse

Die zu berücksichtigenden Schnittgrößen eines Kessels im GuD-Prozeß sind, abgesehen vom Strahlungswärmeverlust, durch den Brennstoff-, evtl. Luft- und Rauchgasstrom und die ein- und austretenden Wasserströme gegeben. Dies können z. B. sein:

- vorzuwärmendes Wasser
- Speisewasser
- Einspritzwasser
- zwischenzuüberhitzender Dampf
- überhitzter Dampf

- zwischenüberhitzter Dampf
- evtl. Heizwasser für Fernwärme

An der Schnittstelle zu anderen Kraftwerkskomponenten muß zusätzlich der Zugriff auf Dampferzeuger-Regelgrößen berücksichtigt werden. Die Regelgrößen sind abhängig vom Wasserumlaufsystem. Sie sind gemäß [55] in Tabelle 7 dargestellt.

Umlaufsystem	Regelgröße
Natur-/Zwangumlauf	Frischdampfdruck Unterdruck im Feuerraum Frischdampftemperatur Trommelwasserstand
Zwangdurchlauf (Sulzer)	Frischdampfdruck Unterdruck im Feuerraum Frischdampftemperatur Wasserstand in Abscheideflasche
Zwangdurchlauf (Benson)	Frischdampfdruck Unterdruck im Feuerraum Frischdampftemperatur Verhältnis Einspritzwasser/Speisewasser

Tabelle 7: Regelgrößen für die unterschiedlichen Wasserumlaufsysteme

Die große Anzahl möglicher Schnittstellen zu anderen Kraftwerkskomponenten wie z. B. Speisewasserbehälter, Speisepumpe oder Dampfturbine bzw. zur Regelung sowie die Vielfalt der Möglichkeiten zur Verschaltung von Einzelkomponenten zu einem Gesamtsystem Dampferzeuger zeigen bereits, daß es nicht sinnvoll ist, bei der Dampferzeugersimulation die verschiedenen Bauarten zu einer gemeinsamen Klasse Dampferzeuger zusammenzufassen. Die oberste Hierarchieebene der Modellierung ist sinnvollerweise die der in Kapitel 4.1 formulierten und eindeutig beschreibbaren Einzelkomponenten wie beispielsweise Feuerraum, Dampferzeugerrohr oder Einspritzkühler. Ist eine *konkrete* Anlage zu simulieren, so besteht die Möglichkeit, eine auf den speziellen Kessel abgestimmte Containerklasse mit entsprechendem inneren Aufbau und zugehörigen Schnittgrößen zu entwickeln.

5 Numerische Betrachtungen

5.1 Bedeutung der Startwerte

Die Simulation komplexer Systeme, wie z. B. das aus mehreren Bilanzierungen und Wärmeübergängen zusammengesetzte Dampferzeugerrohr, hat gezeigt, daß der sinnvollen Initialisierung eine große Bedeutung zukommt. Beispielsweise liefen die Simulationen der Dampferzeugerrohre zum Teil nur zuverlässig bis zum gewünschten Simulationsende, wenn der Verlauf des Druckes, beim Verdampferrohr auch der Dichte, über die Rohrlänge der Strömungsrichtung gemäß abnehmend vorgegeben wurde. Hier kommt es auch darauf an, die Initialisierungswerte nicht allzuweit entfernt von den Werten eines physikalisch möglichen Satzes konsistenter Startwerte vorzugeben. Ansonsten ist der SMILE-Löser darauf angewiesen, solange „an den Variablen zu wackeln", bis eine Stimmigkeit innerhalb des Variablensatzes erreicht worden ist. Die einzelnen Werte liegen zwar in jedem Fall innerhalb der durch die `min`- und `max`-Attribute vorgegebenen Grenzen, dennoch muß der Variablenvektor im ganzen nicht zwangsläufig physikalisch sinnvoll sein. Zwingend vorzugeben sind Werte für Größen, die während der ganzen Simulation unverändert bleiben, z. B. Randbedingungen oder geometrische Abmessungen, sowie Anfangswerte für Differentialgleichungen.

Bei der Vorgabe der Initialisierungswerte innerhalb der Klassenhierarchie ist in jedem Fall die vom SMILE-Löser vorgenommene Initialisierungsreihenfolge zu berücksichtigen. Initialisierungswerte werden in SMILE in der Reihenfolge

- alle `@init`-Blöcke von der obersten Oberklasse angefangen
- auf jeder „Strukturierungsebene":
  - zuerst Initialisierung für die Komponenten (`@component`-Blöcke) und dann
  - Initialisierungen der `@connects`

vergeben. Bei der `@connect`-Verbindung zweier Variablen mit unterschiedlicher Initialisierung sollte immer ein Initialisierungswert mitangegeben werden um sicherzustellen, von welchem der beiden Werte SMILE ausgehen soll, z. B. in den Modellen MassenUndEnergiebilanz:

```
@connect{ c = massbil.c = enerbil.c = 1.0; }
```

Dieser Wert kann in abgeleiteten Klassen für die Initialisierung dort nur noch durch eine zusätzliche `@connect`-Verbindung mit einem entsprechenden Initialisierungswert überschrieben werden, z. B. auf der Ebene der Wasserbilanzierung eines Dampferzeugerrohres. Eine hier stattdessen im `@component`-Block vorgenommene Initialisierung hat keinen Einfluß auf die Startwertermittlung, es wird mit dem in der Klasse MassenUndEnergiebilanz angegebenen Wert initialisiert.

5.2 Stoffwerte im Flüssigkeitsgebiet

Da die Lösungen der Massen- und der Energiebilanzgleichung in der in Kapitel 2.3 hergeleiteten Formulierung die Stoffwerte Dichte ϱ und Enthalpie h liefern, werden sowohl beim Verdampfer- als auch beim Überhitzerrohr die Stoffwertfunktionen für bekannte ϱ und h aufgerufen, d. h. alle anderen Zustandsgrößen sind aus der Dichte und der Enthalpie zu ermitteln.

Beim Verdampferrohr kann nicht ausgeschlossen werden, daß der Löser vorübergehend (z. B. bei der Startwertsuche) auch Zustände im Flüssigkeitsgebiet vermutet, obwohl das Wasser eigentlich verdampft bzw. zumindest siedet. Im Flüssigkeitsgebiet besitzt Wasser jedoch beispielsweise einen sehr steilen Gradienten $\frac{\partial p}{\partial \varrho}$, d. h. wenn die Dichte nur leicht variiert wird, ergeben sich schon sehr große Druckänderungen. Der Löser wird zu keiner Lösung finden. Außerdem stellt der Knick in den Funktionswerten auf der Siedelinie eine Unstetigkeit in der Steigung — und damit evtl. eine Ursache für ständiges Hin- und Herschalten — dar.

Eine Abhilfemöglichkeit besteht in der linearen Extrapolation der Stoffwerte des Naßdampfgebiets auf das Flüssigkeitsgebiet. In Abbildung 13 ist der Übergang vom Flüssig- zum Naßdampfgebiet mit dem Einfluß auf die Druckwerte grafisch dargestellt.

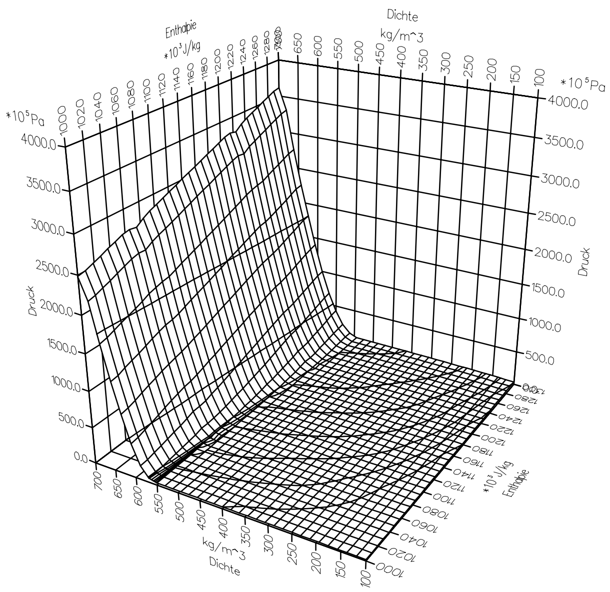

Abbildung 13: Druckverlauf an der Siedelinie in Abhängigkeit von Enthalpie und Dichte

An der Siedelinie ist ein scharfer Knick im Druckverlauf erkennbar. Wird ein kleineres Dichtefenster gewählt (siehe Abbildung 14), so liegt der komplette dargestellte Bereich im Naßdampfgebiet.

Es ist erkennbar, daß der Druckverlauf dort in guter Näherung durch eine Ebenengleichung beschreibbar ist, die dann auch auf den Bereich flüssiger Phase extrapoliert werden kann. Die Ebenengleichung kann jetzt bei Kenntnis eines Zustands 0 und zweier Richtungen $\left(\frac{\Delta p}{\Delta \varrho}\right)_1$ und $\left(\frac{\Delta p}{\Delta h}\right)_2$ leicht ermittelt werden:

$$p\,(h,\varrho) = \left(\frac{\Delta p}{\Delta \varrho}\right)_1 \cdot (\varrho - \varrho_0) + \left(\frac{\Delta p}{\Delta h}\right)_2 \cdot (h - h_0) + p_0 \tag{5.1}$$

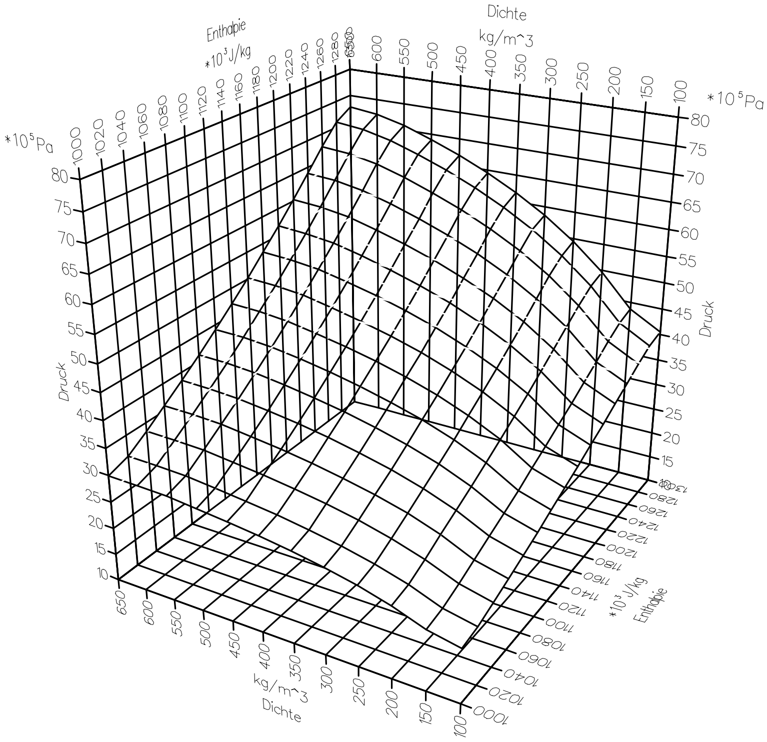

Abbildung 14: Druckverlauf im Naßdampfgebiet in Abhängigkeit von Enthalpie und Dichte

5.3 Algebraische Schleifen

Cellier spricht in [13], S. 25ff, das Problem der „algebraischen Schleifen“ an: Gibt es zwei Variablen (es können auch durchaus mehr Variablen involviert sein), die über algebraische Gleichungen aus der jeweils anderen Größe berechnet werden — $x = f_1(y)$ und $y = f_2(x)$ —, so kann der Löser Probleme bei der Lösung des Gleichungssystems bekommen.

An dem Beispiel der Wärmeübertragung durch Verdampfung seien zwei Möglichkeiten aufgezeigt, diese Schwierigkeiten zu beseitigen:

Für die Bestimmung des Wärmeübergangskoeffizienten α wird eine empirische Gleichung angegeben — Gleichung (2.57): $\alpha = f_1(\dot{q})$. Die Wärmestromdichte $\dot{q}$ selbst hängt aber wiederum über eine algebraische Gleichung von dem Wärmeübergangskoeffizienten ab — Gleichung (2.58): $\dot{q} = f_2(\alpha)$.

Hier führt bereits die in Kapitel 2.4.2.2.2 beschriebene einfache Umformulierung der Gleichungen zum Ziel. Ist dies nicht ohne weiteres möglich, so ist die algebraische Schleife durch Einführung eines künstlichen Integrators „aufzubrechen“. Die Struktur eines solchen Integrators ist in Abbildung 15 aufgezeigt. Bezogen auf das Beispiel der Verdampfung bedeutet dies, z. B. vor

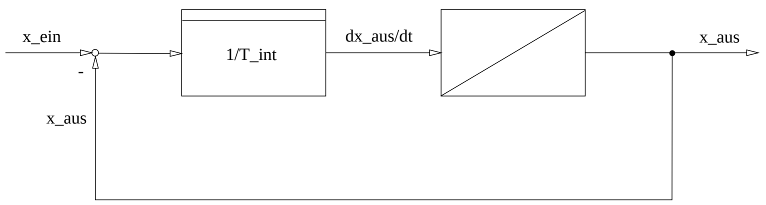

Abbildung 15: Struktur des Integrators

der Berechnung des Wärmeübergangskoeffizienten α einen Integrator zu schalten, der aus der Wärmestromdichte vor Integrator (vI) $\dot{q}_{\mathrm{vI}}$ die Wärmestromdichte $\dot{q}$ selbst berechnet. Das Gleichungssystem lautet jetzt mit den gleichen Funktionen f_1 und f_2: $\alpha = f_1(\dot{q})$, $\dot{q}_{\mathrm{vI}} = f_2(\alpha)$ und $\frac{\partial \dot{q}}{\partial t} = \frac{1}{\tau} \cdot (\dot{q}_{\mathrm{vI}} - \dot{q})$. Hierbei ist die Zeitkonstante τ kleiner als die Zeitkonstante des betrachteten Gesamtsystems zu wählen.

5.4 Normierung

Es kann vorkommen, daß der Löser ein Gleichungssystem mit Variablen stark unterschiedlicher Größenordnung lösen muß. Beträgt das Verhältnis zwischen der Variable mit dem größten und der mit dem kleinsten Betrag etwa 10^{10}, so wird der kleinste Wert als vernachlässigbar klein angesehen und zu Null gesetzt[5]. Daher ist es angebracht, alle Beträge der Variablen in etwa gleiche Größenordnung zu bringen. Das bedeutet beispielsweise, daß der SMILE-Löser große Leistungen nicht in W, sondern in MW zu sehen bekommt, bzw. daß die Angabe von Drücken in bar statt in Pa erfolgt.

Eine Lösungsmöglichkeit besteht darin, die Gleichungen so umzuformulieren, daß Variablen nicht, wie sonst üblich, in SI-Einheiten, sondern mit den entsprechenden normierten Einheiten einzusetzen sind (und zu lösende Größen entsprechend in skalierter Form ermittelt und ausgegeben werden). Die Energiebilanzierung lautet in SMILE beispielsweise bei Skalierung

[5] laut persönlicher Mitteilung von Clemens Klein-Robbehaar vom 20.01.97

von Wärmeleistungen und massenspezifischen Enthalpien

```
@eq diff h[i:1::ND-1] (h[i], h[i-1], c[i], dx, S[i])
{ return (1/dx * c[i] * (h[i-1] - h[i]) * H_NORM + S[i] * Q_NORM) / H_NORM; }
```

statt sonst

```
@eq diff h[i:1::ND-1] (h[i], h[i-1], c[i], dx, S[i])
{ return 1/dx * c[i] * (h[i-1] - h[i])+ S[i]; }
```

Die Simulation eines Verdampferrohres hat gezeigt, daß die Umformulierung aller Gleichungen auf normierte Größen deutlich Rechenzeit einsparen kann: Sowohl die Gesamtzahl der erforderlichen Zeitschritte, als auch der Rechenzeitbedarf konnte auf etwa 2/3 reduziert werden.

Eine elegantere, aber bisher noch nicht realisierte, Lösung besteht darin, bei der Variablendeklaration ein neues Attribut, z. B. „scale", einzuführen, das dafür sorgt, daß die Gleichungen in SI-Einheiten formuliert werden können, der Löser aber die Größen in skalierter Form erhält.

5.5 Rechenzeitbedarf

Die folgenden Simulationen zum Vergleich des Rechenzeitbedarfs bei Variation von Teilmodellen bzw. Anzahl der Diskretisierungselemente wurden ausgeführt an einer Workstation HP 9000/712 (100 MHz), SPECint95 $= 2,89$ und SPECfp95 $= 3,47$.

5.5.1 Vergleich Wärmespeicherung und instationäre Wärmeleitung

Für den Rechenzeitvergleich wird folgendes Beispiel verwendet: Ein Überhitzerrohr — modelliert aus einer Bilanzierung der Dampfseite, einer Bilanzierung der Rauchgasseite, zwei Wärmeübergängen und Wärmespeicherung (ThinWallTubes) bzw. instationärer Wärmeleitung (WaermeleitungRohr) in der Rohrwand — setzt sich in Rohrachsrichtung aus sechs Diskretisierungselementen (ND $= 6$) zusammen. Ausgehend von Startwerten, die von der stationären Lösung nicht allzu weit entfernt sind, wird während einer Simulationszeit von einer Stunde bei einer maximal zulässigen Schrittweite von 60 s die stationäre Lösung gefunden.

In Tabelle 8 ist die Abhängigkeit der benötigten Rechenzeit und die Gesamtzahl der erforderlichen Zeitschritte vom Wandmodell bzw. von der Anzahl der Diskretisierungselemente in rohrwandsenkrechter Richtung (ND_WL) dargestellt.

Es ist keine deutliche Abhängigkeit des Rechenzeitbedarfs von der Anzahl der Rohrwanddiskretisierungselemente erkennbar — die mit wachsender Elementzahl fallende Anzahl der erforderlichen Zeitschritte kompensiert in etwa den steigenden Rechenzeitbedarf pro Zeitschritt. Die Tatsache, daß sich bei Variation der Wandelementzahl weder die Rechenzeit noch die Anzahl der erforderlichen Zeitschritte verändern, legt die Vermutung nahe, daß die Hauptrechenzeit für die Bilanzierung der Fluide und nicht für die der Trennwand benötigt wird. Die durch die Geschwindigkeit der strömenden Medien gegebene kleine Zeitkonstante dominiert das Gesamtsystem gegenüber dem thermisch trägen Verhalten der Rohrwand.

5.5.2 Variation der Anzahl der Rohrelemente in Strömungsrichtung

Variiert man am selben Überhitzerrohr, für die Wand wird hier das einfache Wärmespeichermodell angesetzt, die Anzahl der Diskretisierungselemente in Strömungsrichtung

ND_WL	Modell	Rechen-zeit [s]	Anzahl der Zeitschritte
1	ThinWallTubes	44,5	324
3	WaermeleitungRohr	45,4	192
6	WaermeleitungRohr	38,9	193
10	WaermeleitungRohr	40,4	191
20	WaermeleitungRohr	46,3	185
30	WaermeleitungRohr	44,4	178
50	WaermeleitungRohr	52,2	172

Tabelle 8: Rechenzeitvergleich Wärmespeicherung und instationäre Wärmeleitung

ND, so ist die Abhängigkeit des erforderlichen Rechenzeitbedarfs und die Gesamtzahl der Zeitschritte in Abhängigkeit von der Anzahl der Elemente in Tabelle 9 dargestellt.

ND	Rechen-zeit [s]	Anzahl der Zeitschritte
6	44,5	324
10	87,1	380
20	273,8	455
30	611,1	533

Tabelle 9: Rechenzeitvergleich bei Variation der Elementanzahl in Strömungsrichtung

Es zeigt sich, daß der Rechenzeitbedarf in guter Näherung quadratisch von der Anzahl der Diskretisierungselemente abhängt. In Abbildung 16 ist die Rechenzeit für die Elementzahlen ND $= 6, 10, 20$ und 30 eingezeichnet. Zum Vergleich ist die quadratische Parabel aufgetragen, die sich aus der Minimierung der Fehlerquadrate ergibt.

Es wäre wünschenswert, die Gesamtanzahl der Diskretisierungselemente in Abhängigkeit von den örtlichen Gradienten festzulegen. Das bedeutet jedoch, die Elementanzahl ND — und damit auch die Dimension der die einzelnen Kontrollvolumina beschreibenden Größen — während der Simulation verändern zu können. In SMILE ist das so leider (noch) nicht möglich. Hier ist es erforderlich, am Simulationsbeginn die Elementzahl aufgrund erwarteter Gradienten für die gesamte Simulationsdauer festzulegen.

Edelmann schlägt in [21] vor, für Dampferzeugerrohre ungefähr ND $= 20 \dots 30$ Diskretisierungselemente zu wählen. Bei Rohrlängen von 100 bis 150 m ergeben sich somit Längen der einzelnen Elemente von $\Delta x = 5$ m.

5.5.3 Vergleich stationäre und instationäre Impulsbilanz

Um Aussagen treffen zu können über den Sinn der Verwendung der instationären (Gleichungen (2.25) und (2.26a und b) mit $\tau = 0,01$ s) statt der einfacheren stationären (Gleichungen (2.76a und b) und (2.77)) Impulsbilanzierung, wird ein Überhitzerrohr simuliert. In beiden Fällen der Bilanzformulierung wird nach Ablauf der Simulationszeit die stationäre, nahezu identische, Lösung gefunden. Im stationären Fall wird nur ca. 60% der Rechenzeit des instationären Falles benötigt. Dies ist sicherlich nicht unerheblich auf die gedrittelte Anzahl der benötigten

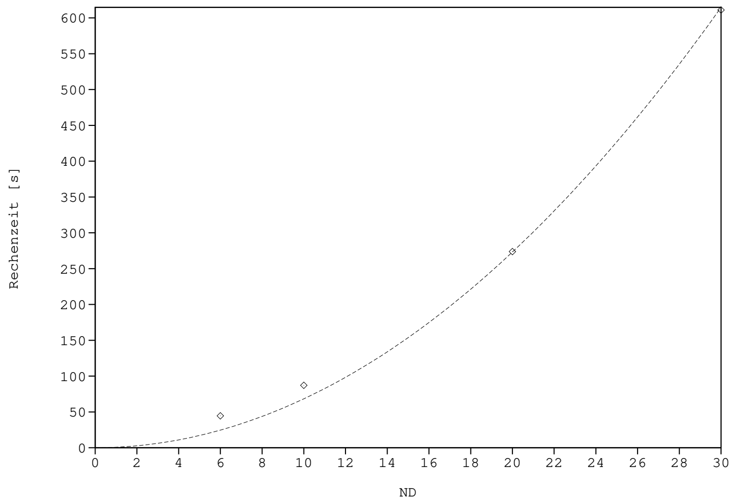

Abbildung 16: Rechenzeitvergleich bei Variation der Elementanzahl in Strömungsrichtung

Zeitschritte zurückzuführen. Hier sollte der Einfluß der Größe von der Zeitkonstante τ nicht vernachlässigt werden.

Interessiert das dynamische Verhalten und nicht nur die stationäre Lösung, so sind noch genauere Vergleiche z. B. mit gemessenen Zeitverläufen von Zustandsgrößen an ausgeführten Dampferzeugern vorzunehmen.

5.5.4 Vergleich reales und ideales Gas am Überhitzerrohr

An einem Überhitzerrohr sollen eventuelle Rechenzeitersparnisse und Rechenungenauigkeiten aufgrund einer Betrachtung von Wasserdampf als ideales Gas aufgezeigt werden.

Simuliert wird ein Überhitzerrohr bestehend aus Rauchgas- und Wasserbilanzierungskomponenten, zwei Wärmeübertragungsmodellen sowie einem Modell zur Beschreibung der Wärmespeicherung in der Rohrwand. Nach einiger Simulationszeit stellt sich ein von den Initialisierungswerten leicht abweichender stationärer Zustand ein.

Es zeigt sich, daß die Betrachtung idealen Gases hier keine Rechenzeit einspart — im Gegenteil: Das mit idealem Gas berechnete Überhitzerrohr erfordert etwa 40% mehr Rechenzeit als das mit realem Gas. Hier konnten auch nicht allzu große Ersparnisse erwartet werden, da nach wie vor zwei Stoffwertfunktionsaufrufe vorgenommen werden müssen: Statt beim realen Wasserdampf Druck und Temperatur aus zwei `whd`-Funktionsauswertungen zu ermitteln, müssen beim idealen Gas die Größen Siedetemperatur und Sattdampfenthalpie bestimmt werden (siehe Gleichung 4.19). Die Sättigungsgrößen werden iterativ aus der Phasengleichgewichtsbedingung bestimmt und sind damit relativ rechenzeitaufwendig.

Da das Wasser im Sattdampfzustand in das Rohr eintritt — hier ist die vereinfachende Betrachtung als ideales Gas sehr grob —, ergibt sich auch eine spürbare Abweichung im resultierenden Frischdampfzustand: Statt bei der Betrachtung realen Gases 58,9 bar/504 °C erhält man bei der vereinfachten Betrachtungsweise einen Frischdampfzustand von 66,7 bar/511 °C. Das Rauchgas kühlt sich hierbei nur auf 462 °C anstatt auf 445 °C ab. Bei Eintritt des Wasserdampfes im stärker überhitzten Bereich sind hier deutlich bessere Ergebnisse zu erwarten.

6 Zusammenfassung und Ausblick

Dynamische Rechenprogramme liefern einen wichtigen Beitrag zur Beschreibung des instationären Verhaltens energietechnischer Anlagen. Inhalt dieser Arbeit ist die Modellierung grundlegender Elemente von Dampferzeugern in der Simulationsumgebung SMILE. Nachzubilden ist nicht eine konkrete Anlage, sondern es sind Einzelkomponenten von Dampferzeugern zu erstellen, die als Bestandteil einer ausbaufähigen objektorientierten Modellbibliothek in verschiedenen Anlagenkonfigurationen Anwendung finden können.

Zunächst wurden als Grundlage der zu betrachtenden Dampferzeugerkomponenten Bilanzierungsgleichungen für durchströmte Kontrollvolumen formuliert und implementiert. Hier sind bereits unter Annahme entsprechender Vereinfachungen unterschiedliche Anwendungsbereiche, wie z. B. verschiedene Umlaufsysteme, berücksichtigt worden. Eine geeignete Strukturierung der Basisklassen erlaubt die leichte Austauschbarkeit von Teilmodellen zur Anpassung der Modellierungstiefe an den Umfang des betrachteten Gesamtsystems. Besonders in den unterschiedlichen Wärmespeicherungs- und -übertragungsbeziehungen garantieren gemeinsame Schnittstellen eine flexible Handhabung bei deren Einbindung in die Modelle konkreter Dampferzeugerbauteile.

Bei geeigneter Implementation vorhandener Stoffwertfunktionen konnten mit Hilfe der Basisklassen bereits Einzelbestandteile von Dampferzeugern nachgebildet werden. Eine Vereinheitlichung der Modellierung der Richtung des Informationsflusses liefert zunächst eine Hilfe zur Vermeidung von Verknüpfungsproblemen an den Schnittstellen. Unter Berücksichtigung dieses Modellierungsprinzips sind die Einzelkomponenten Feuerraum, beheiztes Dampferzeugerrohr und Einspritzkühler beschrieben worden. Das Dampferzeugerrohr in Form des Überhitzer-, Verdampfer- und Ecorohres wurde trotz unterschiedlicher Bilanzierungs- und Wärmeübertragungsbeziehungen unter Wahrung gemeinsamer Schnittstellen und Informationsflußrichtungen zur einheitlichen Einbindung in das Gesamtsystem modelliert. Zur Nachbildung eines kompletten Dampferzeugers sind in Zukunft noch Komponenten wie z. B. Dampftrommel, Umwälzpumpe oder Verdampferrohr im Naturumlaufbetrieb zu modellieren. Es ist noch der Nachweis zu erbringen, daß die bereits modellierten Einzelkomponenten des Dampfkessels auch in einem komplexen Gesamtsystem Dampferzeuger ihre Funktion erfüllen.

Einige numerische Betrachtungen beschreiben mit der Dampferzeugermodellierung verknüpfte Schwierigkeiten. Sie liefern demgegenüber aber auch Erkenntnisse über Möglichkeiten der Rechenzeiteinsparung.

7 Tabellenverzeichnis

Tabellenverzeichnis

8 Abbildungsverzeichnis

Abbildungsverzeichnis

9 Literaturverzeichnis

Literatur

[1] Ahrens, Klaus ; Fischer, Joachim: *Objektorientierte Programmierung.* 1. Aufl. Berlin : Verlag Technik, 1992. – ISBN 3-341-01062-9.

[2] Alisch, Rolf-Stefan: *Engineeringsimulator für die Analyse der Dynamik des Gas- und Dampfturbinenprozesses mit einem Zwangdurchlaufdampferzeuger.* Stuttgart : Selbstverlag des Institutes für Verfahrenstechnik und Dampfkesselwesen der Universität Stuttgart, 1991. – ISBN 3-928123-04-1.

[3] Baehr, Hans Dieter: *Thermodynamik.* 9. Aufl. Berlin : Springer-Verlag, 1996. – ISBN 3-540-60157-0.

[4] Baehr, Reinhart ; Blangetti, Francisco L. ; Braun, Bruno ; Casper, William ; Haase, Helmut ; Häfele, Carl Heinz ; Huschauer, Helmut ; Jung, Manfred ; Krane, Rolf ; Kriener, Albert ; Lang, Helmut ; Martini, Manfred ; Oude-Hengel, Hans-Herbert ; Sauer, Erich ; Schwarzenbach, Alfred ; Schüller, Karl-Heinz ; Zimmermann, Peter: *Konzeption und Aufbau von Dampfkraftwerken.* Bd. 5. 1. Aufl. Köln : Technischer Verlag Resch, 1985 (Handbuchreihe Energie). – ISBN 3-87806-085-8.

[5] Beitz, Wolfgang ; Küttner, Karl-Heinz (Hrsg.): *Dubbel — Taschenbuch für den Maschinenbau.* 17. Aufl. Berlin : Springer-Verlag, 1990. – ISBN 3-540-52381-2.

[6] Berndt, Georg: *Mathematisches Modell eines Naturumlaufdampferzeugers zur Störfallsimulation und dessen experimentelle Untersuchung.* Universität Stuttgart, Dissertation, 1984.

[7] Biersack, Maya ; Klein-Robbehaar, Clemens ; Klose, Matthias ; Lutz, Sven ; Simons, Martin: Smile: Tutorial und Referenz Handbuch Version 0.8 / Technische Universität Berlin and GMD FIRST. 1995.

[8] Bohn, Thomas ; Bitterlich, Walter: *Grundlagen der Energie- und Kraftwerkstechnik.* Bd. 1. 1. Aufl. Köln : Technischer Verlag Resch, 1982 (Handbuchreihe Energie). – ISBN 3-87806-054-8.

[9] Booch, Grady: *Objektorientierte Analyse und Design.* 1. Aufl. Bonn : Addison-Wesley, 1994. – ISBN 3-89319-673-0.

[10] Brandt, F.: *Wärmeübertragung in Dampferzeugern und Wärmeaustauschern.* Essen : Vulkan-Verlag, 1985. – ISBN 3-8027-2274-4.

[11] Bunzemeier, Andreas: *Mathematisches Modell zur regeldynamischen Analyse eines Dampferzeugers mit zirkulierender Wirbelschichtfeuerung.* Düsseldorf : VDI-Verlag GmbH, 1992. – ISBN 3-18-147306-5.

[12] Buxmann, Joachim: Vorlesungsskript Dampferzeuger / TU Hamburg-Harburg. 1988.

[13] Cellier, François E.: *Continuous System Modeling.* 1. Aufl. New York : Springer-Verlag, 1991. – ISBN 0-387-97502-0.

[14] Chawla, J. M. ; Wiskot, Günter: *Wärmeübertragung — Berechnungen mit dem PC.* 1. Aufl. Düsseldorf : VDI-Verlag GmbH, 1992. – ISBN 3-18-400949-1.

[15] Doležal, R. ; Hönig, O. ; Hübner, H. ; Kammer, G. v. d.: Nichtlineares Modell eines Dampferzeugers. In: *VDI-Bericht* 276 (1977), S. 145–151.

[16] DOLEŽAL, R. ; ROLF, A.: Nichtlineare Simulation von dynamischen Vorgängen mit und ohne Phasenumwandlung in Oberflächen-Wärmetauschern aller Art. In: *Regelungstechnik* 29 (1981), Nr. 9, S. 312–318.

[17] DOLEŽAL, R. ; v. d. KAMMER, G. ; KÖNIGSDORF, E.: Simulation des Anfahrens eines Dampferzeugers mit einem universalen, für große Zustandsübergänge geeigneten Modell. In: *VGB Kraftwerkstechnik* 55 (1975), Nr. 8, S. 506–513.

[18] DOLEŽAL, Richard: Entwicklung des Modells des Dampferzeugers. In: *VDI-Bericht* 276 (1977), S. 51–59.

[19] DOLEŽAL, Richard: *Dampferzeugung — Verbrennung, Feuerung, Dampferzeuger.* Berlin : Springer-Verlag, 1990. – ISBN 3-540-13771-8.

[20] DYMEK, Thomas Georg: *Modulares und bedienerfreundliches Rechenprogramm für die Kraftwerksdynamik.* Düsseldorf : VDI-Verlag GmbH, 1991. – ISBN 3-18-146006-0.

[21] EDELMANN, Heiner: *Modellierung der Dynamik und des Regelverhaltens für einen Dampferzeuger mit zirkulierender Wirbelschichtfeuerung.* Düsseldorf : VDI-Verlag GmbH, 1992. – ISBN 3-18-147506-8.

[22] EFFENBERGER, Helmut: *Dampferzeuger.* 2. Aufl. Leipzig : VEB Deutscher Verlag für Grundstoffindustrie, 1989. – ISBN 3-342-00376-6.

[23] EIRUND, Helmut: *Objektorientierte Programmierung.* 1. Aufl. Stuttgart : B. G. Teubner, 1993. – ISBN 3-519-02938-3.

[24] ENGEL, Reinhard: *Objektorientierte Programmierung: eine Einführung.* 3. Aufl. Haar bei München : Markt & Technik, 1990. – ISBN 3-89090-353-3.

[25] FRENSCH, J. ; KLEFENZ, G.: Die Dynamik der Dampferzeugung im Bensonkessel. In: *Brennstoff–Wärme–Kraft* 13 (1961), S. 532–537.

[26] FRITZ, Walter: Grundlagen der Wärmeübertragung beim Verdampfen von Flüssigkeiten. In: *Chem. Ing. Tech.* 35 (1963), Nr. 11, S. 753–764.

[27] GNIELINSKI, V.: *Wärmeübergang bei erzwungener einphasiger Strömung.* In: *VDI-Wärmeatlas — Berechnungsblätter für den Wärmeübergang*, Kapitel G. 7. Aufl. Düsseldorf : VDI-Verlag GmbH, 1994.

[28] GORENFLO, D.: *Behältersieden (Sieden bei freier Konvektion).* In: *VDI-Wärmeatlas — Berechnungsblätter für den Wärmeübergang*, Kapitel Hab. 7. Aufl. Düsseldorf : VDI-Verlag GmbH, 1994.

[29] GREGORIG, Romano: *Wärmeaustausch und Wärmeaustauscher.* 1. Aufl. Aarau : Sauerländer, 1973. – ISBN 3-7941-0132-4.

[30] GRÖBER, H. ; ERK, S. ; GRIGULL, Ulrich: *Die Grundgesetze der Wärmeübertragung.* 3. Aufl. Berlin : Springer-Verlag, 1988. – ISBN 3-540-02982-6.

[31] GROSSE-DUNCKER, Ernst: *Berechnung und Analyse des Anfahrvorganges eines Naturumlaufdampferzeugers mit einem physikalisch-mathematischen Modell.* Düsseldorf : VDI-Verlag GmbH, 1987. – ISBN 3-18-149706-1.

[32] HEITMÜLLER, Ralf J.: *Mathematische Simulation des dynamischen Verhaltens eines Zwangdurchlaufdampferzeugers beim Durchfahren des kritischen Punktes.* Düsseldorf : VDI-Verlag GmbH, 1987. – ISBN 3-18-149906-4.

[33] Hell, Friedrich: *Grundlagen der Wärmeübertragung.* 3. Aufl. Düsseldorf : VDI-Verlag GmbH, 1982. – ISBN 3-18-400529-1.

[34] Hönig, O.: *Untersuchung des dynamischen Verhaltens eines konvektiv beheizten Naturumlauf-Dampferzeugers mit einem mathematischen Modell.* Düsseldorf : VDI-Verlag GmbH, 1980.

[35] Kernighan, Brian W. ; Ritchie, Dennis M.: *Programmieren in C.* 2. Aufl. München : Carl Hanser Verlag, 1990. – ISBN 3-446-15497-3.

[36] Klefenz, G.: Regelungsdynamische Untersuchung eines Bensonkessels. In: *Brennstoff–Wärme–Kraft* 17 (1965), Nr. 11, S. 532–540.

[37] Klefenz, Günter: *Die Regelung von Dampfkraftwerken.* 4. Aufl. Mannheim : Wissenschaftsverlag, 1991. – ISBN 3-411-15284-2.

[38] Klug, Manfred: *Simulation von Störfällen in Zwangdurchlaufdampferzeugern mit einem physikalisch-mathematischen Modell.* Düsseldorf : VDI-Verlag GmbH, 1984. – ISBN 3-18-144206-2.

[39] Konakow, P. K.: Wärmeübertragung in Kesselfeuerungen. In: *Brennstoff, Wärme, Kraft* 7 (1955), Nr. 3, S. 132f.

[40] Ledinegg, Maximilian: *Dampferzeugung — Dampfkessel, Feuerungen.* 2. Aufl. Wien : Springer-Verlag, 1966.

[41] Leithner, Reinhard: Dynamik im Großdampferzeugerbau. In: *Elektrizitätswirtschaft* 79 (1980), Nr. 8, S. 281–290.

[42] Leithner, Reinhard ; Linzer, Vladimir: *Einfaches Dampferzeugermodell (digitale Simulation).* Düsseldorf : VDI-Verlag GmbH, 1975. – ISBN 3-18-144106-6.

[43] Martin, H.: *Instationäre Wärmeleitung in ruhenden Körpern.* In: *VDI-Wärmeatlas — Berechnungsblätter für den Wärmeübergang*, Kapitel Ec. 7. Aufl. Düsseldorf : VDI-Verlag GmbH, 1994.

[44] Mayinger, F. ; Reineke, H.-H. ; Schramm, R. ; Steinmetz, P.: Ein Rechenprogramm zur nichtlinearen Simulation der Dynamik von Benson-Dampferzeugern. In: *Brennstoff–Wärme–Kraft* 30 (1978), Nr. 8, S. 329–333.

[45] Patankar, Suhas V.: *Numerical Heat Transfer and Fluid Flow.* 1. Aufl. Washington : Hemisphere Publishing Corporation, 1980. – ISBN 0-07-048740-5.

[46] Rohse, Hartmut: *Untersuchung der Vorgänge beim Übergang vom Umwälz- zum Zwangdurchlaufbetrieb mit einer dynamischen Dampferzeugersimulation.* Düsseldorf : VDI-Verlag GmbH, 1995. – ISBN 3-18-332706-6.

[47] Rösel, G. ; Röder, H.: Ermittlung eines nichtlinearen Modells für große Dampferzeuger. In: *Messen, steuern, regeln* 21 (1978), Nr. 2, S. 73–77.

[48] Schlünder, Ernst-Ulrich: *Einführung in die Wärmeübertragung.* 7. Aufl. Braunschweig : Friedr. Vieweg & Sohn, 1991. – ISBN 3-528-63314-X.

[49] Schmidt, Ernst (Hrsg.): *Zustandsgrößen von Wasser und Wasserdampf in SI-Einheiten.* 4. Aufl. Berlin : Springer-Verlag, 1989. – ISBN 3-540-09601-9.

[50] Schöler, Jürgen: *Ein Gesamtmodell für Dampferzeugeranlagen mit zirkulierender Wirbelschichtfeuerung.* Aachen : Verlag Shaker, 1993. – ISBN 3-86111-362-7.

[51] SCHUMACHER, A. ; WALDMANN, H.: *Wärme- und Strömungstechnik im Dampferzeugerbau — Grundlagen und Berechnungsverfahren.* 1. Aufl. Essen : Vulkan-Verlag, 1972. – ISBN 3-8027-2476-3.

[52] SCHUMACHER, Ulrich: *Über das instationäre Verhalten des Wasser/Dampfsystems eines Heizkraftwerks.* RWTH Aachen, Dissertation, 1989.

[53] SMITH, David N.: *Konzepte der objektorientierten Programmierung.* 1. Aufl. Hamburg : McGraw-Hill, 1993. – ISBN 3-89028-401-9.

[54] STEPHAN, Karl ; MAYINGER, Franz: *Thermodynamik, Band 1: Einstoffsysteme, Grundlagen und technische Anwendungen.* 14. Aufl. Berlin : Springer-Verlag, 1992.

[55] SÜLTMANN, Jana: *Entwurf und Implementation eines Regelungskonzeptes für die objektorientierte Programmiersprache SMILE.* Technische Universität Hamburg-Harburg, Diplomarbeit, 1997.

[56] THOMANN, Walter: *Ein nichtlineares mathematisches Modell hoher Genauigkeit für Dampferzeuger.* Düsseldorf : VDI-Verlag GmbH, 1991. – ISBN 3-18-144806-0.

[57] THUMM, Carl Thomas: *Wirkleistungs-Sekundenreserve-Maßnahmen, untersucht am Beispiel eines Dampfkraftwerksblocks.* Universität Stuttgart, Dissertation, 1989.

[58] TUMMESCHEIT, Hubertus: *Implementation des Programmsystems SMILE zur Simulation energietechnischer Prozesse und Konzeption einer objektorientierten Modellbibliothek.* Technische Universität Hamburg-Harburg, Diplomarbeit, 1996.

[59] VEREIN DEUTSCHER INGENIEURE, VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen (GVC) (Hrsg.): *VDI-Wärmeatlas — Berechnungsblätter für den Wärmeübergang.* 7. Aufl. Düsseldorf : VDI-Verlag GmbH, 1994. – ISBN 3-18-401362-6.

[60] VGB TECHNISCHE VEREINIGUNG DER GROSSKRAFTWERKSBETREIBER E. V.(Hrsg.): *Fachkunde für den Dampfkraftwerksbetrieb — Dampferzeugung 1.* 2. Aufl. Essen : Richard Bacht, 1971.

[61] VGB TECHNISCHE VEREINIGUNG DER GROSSKRAFTWERKSBETREIBER E. V.(Hrsg.): *Fachkunde für den Dampfkraftwerksbetrieb — Dampferzeugung 2.* 2. Aufl. Essen : A. Sutter, 1975.

[62] VORTMEYER, D.: *Strahlung technischer Oberflächen.* In: *VDI-Wärmeatlas — Berechnungsblätter für den Wärmeübergang*, Kapitel Ka. 7. Aufl. Düsseldorf : VDI-Verlag GmbH, 1994.

[63] WAGNER, Walter: *Thermische Apparate und Dampferzeuger — Planung und Berechnung.* 2. Aufl. Würzburg : Vogel Verlag, 1990. – ISBN 3-8023-0157-9.

[64] WITT, Kurt-Ulrich: *Einführung in die objektorientierte Programmierung.* 1. Aufl. München : R. Oldenbourg Verlag, 1992. – ISBN 3-486-21615-5.

[65] WITTE, Ulrich (Hrsg.): *Dampferzeugertechnik.* 25. Aufl. Essen : Vulkan-Verlag, 1992. – ISBN 3-8027-2510-7.

Anhang

A Erläuterungen zur Notation von Klassenstruktur und `@export`-Variablen

Im Anhang B ist der in SMILE implementierte Programmcode aufgeführt. Abschnittsweise sind zunächst die Vererbungsstruktur grafisch und die jeweiligen `@export`-Variablen tabellarisch dargestellt. Die dafür verwendete Notation soll kurz erläutert werden.

A.1 Wolkenbilder

Im Anhang B wird in der von Booch in [9] vorgeschlagenen Notation die Klassenstruktur grafisch dargestellt. Sie beinhaltet zwei mögliche Beziehungen der in gestrichelten Wolken dargestellten Instanzen von Klassen (siehe Abbildung 17):

- „is-a"-Beziehung: Die Vererbung von Klassen wird durch einen Pfeil, der auf die Elternklasse zeigt, dargestellt.
- „part-of"-Beziehung: Die Aggregration von Klassen wird durch eine Linie mit einem ausgefüllten Kreis auf der Seite der zusammenfassenden Klasse verbildlicht.

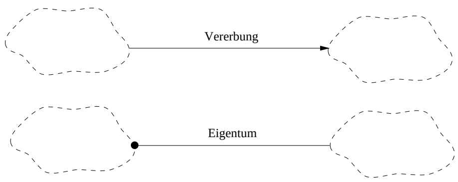

Abbildung 17: Booch-Notation

A.2 Tabellierung der `@export`-Variablen

In der Simulationsumgebung SMILE werden bei Variablen zwei Sichtbarkeitsspezifikationen unterschieden: Während in einer Klasse deklarierte `@protected`-Größen nur in der Klasse selbst und davon abgeleiteten Klassen bekannt sind, sind die `@export`-Variablen für die Verschaltung fertiger Komponenten von Bedeutung — sie sind nach außen sichtbar und können über `@connects` mit anderen `@export`-Variablen verbunden werden. Es ist darauf zu achten, daß durch `@connects` verknüpfte `@export`-Größen durch höchstens eine Gleichung definiert sind.

Im Anhang B sind die `@export`-Variablen der Klassen tabellarisch dargestellt. Vererbte `@export`-Größen sind nicht wiederholt aufgeführt (es sei denn, sie werden mit dem Gleichungsattribut `eq` redeklariert), sie können in der jeweiligen Elternklasse nachgesehen werden. Vektorwertige Variablen werden durch eckige Klammern gekennzeichnet, der Wert in der Klammer gibt die Dimension an. In der Tabellenspalte „Gleichung" sind durch eine Gleichung definierte Größen mit einem $\times$ gekennzeichnet. Somit läßt sich leicht die Richtung des Informationsflusses der jeweiligen Klasse erkennen, und unzulässige `@connects` können vermieden werden.

B Klassenstruktur, @export-Variablen und Programmcode

B.1 Headerdateien

B.1.1 „arraydefs.h“

```
#define ND    6
#define ND_0  6
#define ND_1  5
#define ND_2  4

#define ND_WL   4
#define ND_WL_1 3
```

B.1.2 „consts.h“

```
#define PI      3.1415926
#define PKRIT 220.64e+05
```

B.1.3 „zeitkonstanten.h“

```
#define TAU_KLEIN   0.0001
#define TAU_MITTEL  0.001
#define TAU_GROSS   0.01
```

B.1.4 „normierung.h“

```
#define Q_NORM  1.0e+06         // Waermeleistung in MW
#define H_NORM  1.0e+03         // spezifische Enthalpie in kJ/kg
#define P_NORM  1.0e+05         // Absolutdruck in bar
#define S_NORM  1.0e+05         // spezifische Entropie in kJ/(kg*K)
```

B.2 Bilanzierungsklassen

Die Klassen der Energiebilanzen und der Massenbilanzierung für kompressible Medien sowie der Wärme- (Teil von Anhang B.4) und Massespeicherung (Anhang B.5) basieren auf der Modellierung von Hubertus Tummescheit in [58]. Nachdem er sie überarbeitet und freundlicherweise zur Verfügung gestellt hat, wurden im Rahmen dieser Arbeit zum Teil auch Änderungen vorgenommen.

Klassenname	Elternklasse	@export-Variable	Gleichung
BalanceBaseClass	SmileObject	N	
		length	
		aquer	
		dx	×
		vol	×
BalanceBaseClassCM	BalanceBaseClass	c	
		rho	

Klassenname	Elternklasse	`@export`-Variable	Gleichung
BalanceBaseClassDM	BalanceBaseClass	`c[ND+1]`	
		`rho[ND]`	
ConcentratedMassBalance	BalanceBaseClassCM	`m_dot_in`	
		`m_dot_out`	
		`mass[ND]`	×
		`rho`	×
IncompressibleMassBalance	BalanceBaseClassCM	`m_dot_in`	
		`m_dot_out`	×
		`mass[ND]`	×
		`rho`	
		`c`	×
DistributedMassBalance	BalanceBaseClassDM	`m_dot[ND+1]`	
		`mass[ND]`	×
		`rho[ND]`	×
ConvectiveTemperatureTransportCM	BalanceBaseClassCM	`T_in`	
		`T_out`	×
		`T[ND]`	×
		`Q_dot[ND]`	
		`cp`	
ConvectiveEnthalpyTransportCM	BalanceBaseClassCM	`h_in`	
		`h_out`	×
		`h[ND]`	×
		`T[ND]`	
		`Q_dot[ND]`	
		`cp`	
ConvectiveTemperatureTransportDM	BalanceBaseClassDM	`T_in`	
		`T_out`	×
		`T[ND]`	×
		`Q_dot[ND]`	
		`cp`	
ConvectiveEnthalpyTransportDM	BalanceBaseClassDM	`h_in`	
		`h_out`	×
		`h[ND]`	×
		`Q_dot[ND]`	
		`cp`	
ImpulsbilanzInkompressibel	BalanceBaseClassCM	`p_in`	×
		`p_out`	
		`di`	
ImpulsbilanzStationaer	BalanceBaseClassDM	`m_dot_in`	
		`m_dot_out`	
		`m_dot[ND+1]`	×
		`c[ND+1]`	×
		`p[ND]`	
		`rho_in`	
		`di`	
ImpulsbilanzInstationaer	BalanceBaseClassDM	`m_dot_in`	
		`m_dot_out`	

Klassenname	Elternklasse	@export-Variable	Gleichung
		m_dot[ND+1]	×
		c[ND+1]	×
		p[ND]	
		rho_in	
		di	
ImpulsbilanzInstationaer2Ph	ImpulsbilanzInstationaer	x[ND]	
RgEnergiebilanz	ConvectiveTemperature-TransportCM	aquer	
		rho	
		mass[ND]	×
		c	×
		m_dot	
		p	
		h[ND]	×
MassenUndEnergiebilanzEnthalpieCM	BalanceBaseClassCM	T[ND]	
		h[ND]	
		Q_dot[ND]	
		m_dot_in	
		m_dot_out	
		rho	
		h_in	
		h_out	
		cp	
MassenUndEnergiebilanz-EnthalpieRohrCM	MassenUndEnergiebilanz-EnthalpieCM	aquer	×
MassenUndEnergiebilanzEnthalpieDM	BalanceBaseClassDM	h[ND]	
		Q_dot[ND]	
		m_dot[ND+1]	
		rho[ND]	
		h_in	
		h_out	
		cp	
MassenUndEnergiebilanz-EnthalpieRohrDM	MassenUndEnergiebilanz-EnthalpieDM	aquer	×
MassenUndEnergiebilanzTemperaturCM	BalanceBaseClassCM	T[ND]	
		Q_dot[ND]	
		m_dot_in	
		m_dot_out	
		rho	
		T_in	
		T_out	
		cp	
MassenUndEnergiebilanz-TemperaturRohrCM	MassenUndEnergiebilanz-TemperaturCM	aquer	×
MassenUndEnergiebilanzTemperaturDM	BalanceBaseClassDM	T[ND]	
		Q_dot[ND]	
		m_dot[ND+1]	

Klassenname	Elternklasse	@export-Variable	Gleichung
		rho[ND] T_in T_out cp	
MassenUndEnergiebilanz-TemperaturRohrDM	MassenUndEnergiebilanz-TemperaturDM	aquer	×
MassenUndEnergiebilanzEnthalpie-Inkompressibel	BalanceBaseClassCM	T[ND] h[ND] Q_dot[ND] m_dot_in m_dot_out rho h_in h_out cp	×
MassenUndEnergiebilanzEnthalpie-InkompressibelRohr	MassenUndEnergiebilanzEnthalpie-Inkompressibel	aquer	×
MassenUndEnergiebilanzTemperatur-Inkompressibel	BalanceBaseClassCM	T[ND] Q_dot[ND] m_dot_in m_dot_out rho T_in T_out cp	
MassenUndEnergiebilanz-TemperaturInkompressibelRohr	MassenUndEnergiebilanz-TemperaturInkompressibel	aquer	×
WasserbilanzCM_cuts	BalanceBaseClassCM	T[ND] Q_dot[ND] h_in p_in rho_in m_dot_in h_out p_out m_dot_out	
WasserbilanzDM_cuts	BalanceBaseClassDM	T[ND] Q_dot[ND] h_in p_in rho_in m_dot_in h_out p_out m_dot_out	
WasserbilanzEcorohr	WasserbilanzCM_cuts		

Klassenname	Elternklasse	@export-Variable	Gleichung
WasserbilanzVerdampferrohr	WasserbilanzDM_cuts	di	
WasserbilanzUeberhitzerrohr	WasserbilanzDM_cuts	di	

Tabelle 10: @export-Variablen der Bilanzierungsklassen

B.2.1 „mass.smi“

Die mathematische Modellierung der Massenbilanzierungsklassen ist in Kapitel 2.3.1 ab Seite 6 zu finden. Die Gleichungen für den darin enthaltenen Reibungsdruckabfall stehen in Kapitel 2.6 ab Seite 19.

```
#include "arraydefs.h"
#include "mol.h"
#include "energy.smd"

@interface ConcentratedMassBalance : BalanceBaseClassCM
{
  @export //Fluidin
    double m_dot_in [doc: "Massenstrom", min: 0.0, max: 1.0e6, unit: "kg/s"];

  @export //Fluidout
    double m_dot_out [doc: "Massenstrom", min: 0.0, max: 1.0e6, unit: "kg/s"];

  @export //massbalance
    double mass[ND] [eq, doc: "mass", min: 0.0, max: 1.0e5, unit: "kg"];
    double rho [eq, doc: "density", min: 0.0, max: 25.0e3, unit: "kg/m^3"];
}

  @init {
      m_dot_in = m_dot_out = 0.0;
      rho =  1000.0;
      }

@end

@implementation ConcentratedMassBalance

  @eq diff rho (m_dot_in, m_dot_out, vol, N) {
        return  (m_dot_in + m_dot_out) / (vol * N); }

  @eq cont mass[i:0::ND-1] (rho, vol) {
        return rho * vol; }

@end




@interface DistributedMassBalance : BalanceBaseClassDM
{
  @export //properties
    double m_dot[ND+1] [doc: "massflow", min: "0.0", max: "1.0e4", unit: "kg/s"];

  @export //massbalance
    double mass[ND] [eq, doc: "mass", min: 0.0, max: 1.0e5, unit: "kg"];
```

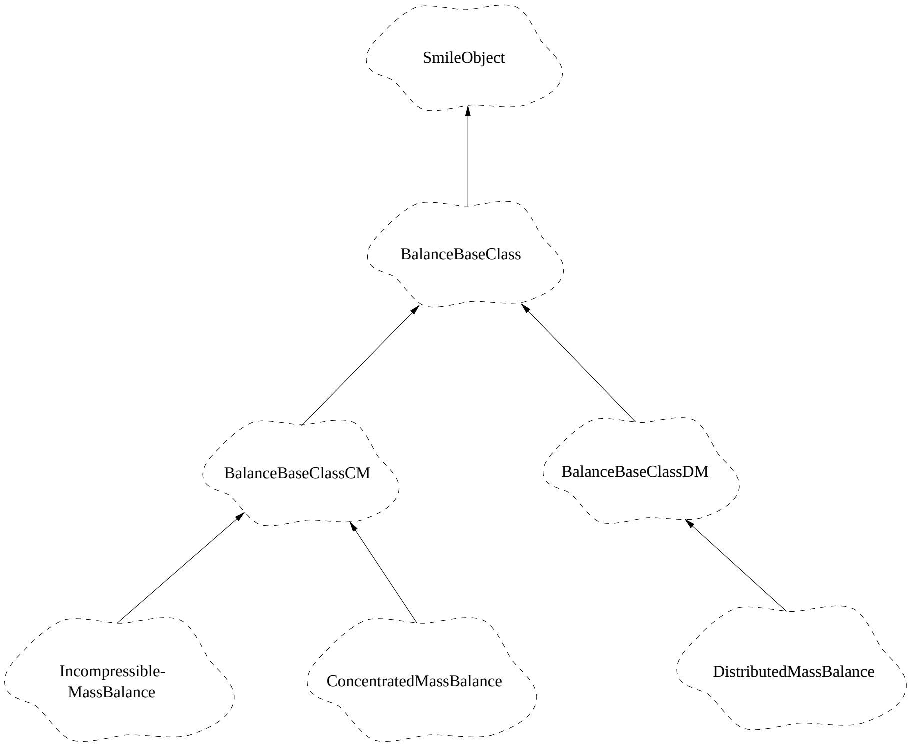

Abbildung 18: Klassenstruktur Massenbilanzierung

```
    double rho[ND] [eq, doc: "density", min: 0.0, max: 25.0e3, unit: "kg/m^3"];
}
@end

@implementation  DistributedMassBalance

        @eq diff rho[i:0::ND-1] (vol, m_dot[i], m_dot[i+1]) //Massenbilanz
        { return (m_dot[i] - m_dot[i+1]) / vol; }

        @eq cont mass[i:0::ND-1] (rho[i], vol) {
        return rho[i] * vol; }
@end



@interface IncompressibleMassBalance : BalanceBaseClassCM{

  @export //Fluidin
    double m_dot_in  [doc: "Massenstrom", min: 0.0, max: 1.0e6, unit: "kg/s"];

  @export //Fluidout
    double m_dot_out [eq, doc: "Massenstrom", min: 0.0, max: 1.0e6, unit: "kg/s"];

  @export //massbalance
    double mass[ND]  [eq, doc: "mass", min: 0.0, max: 1.0e5, unit: "kg"];
    double rho       [doc: "density", min: 0.0, max: 25.0e3, unit: "kg/m^3"];
    double c         [eq, doc: "speed", min: 0.0, max: 1.0e+03, unit: "m/s"];
}

  @init {
      m_dot_in = m_dot_out = 0.0;
```

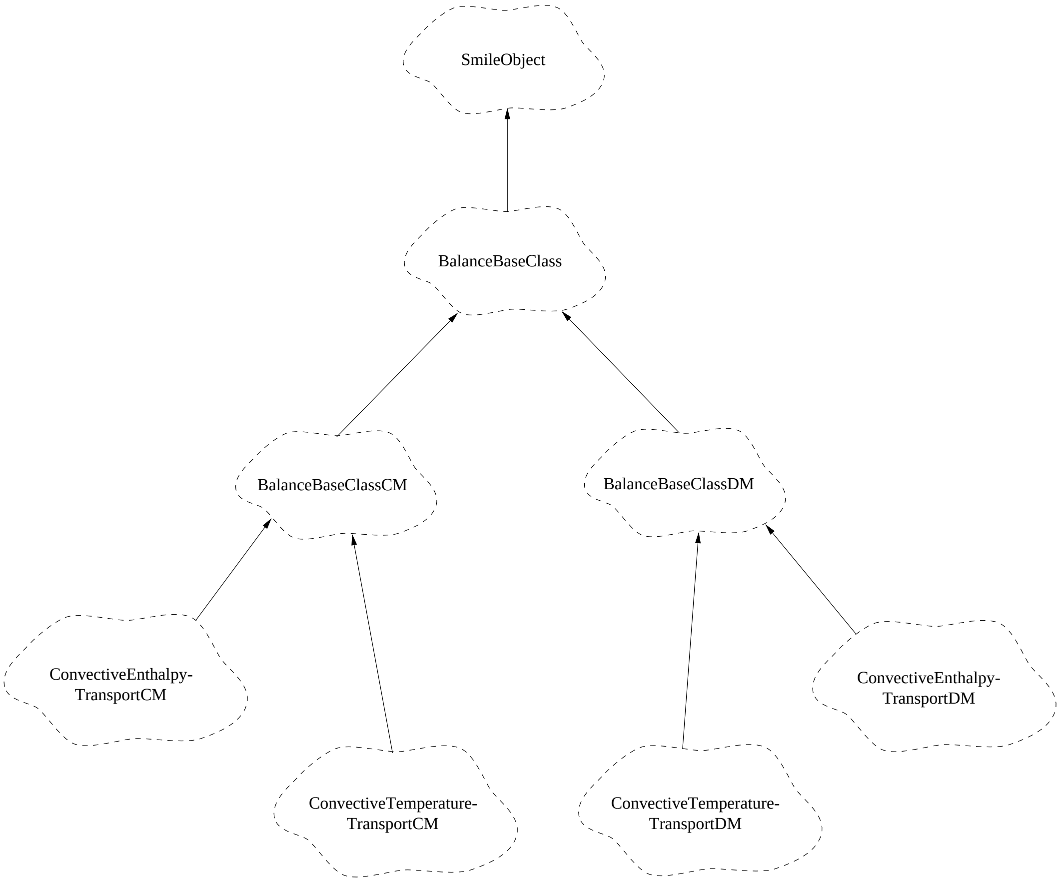

Abbildung 19: Klassenstruktur Energiebilanzierung

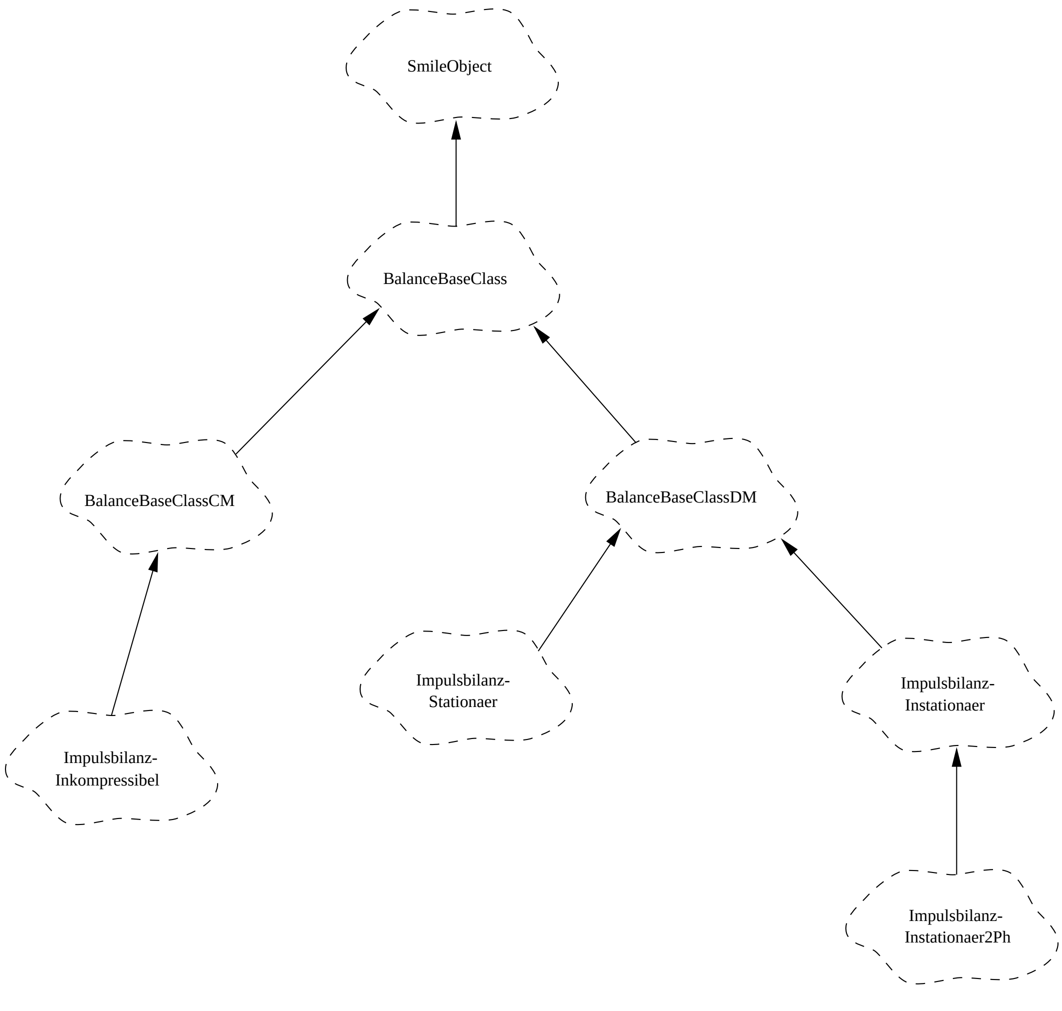

Abbildung 20: Klassenstruktur Impulsbilanzierung

```
      rho =  1000.0;
      }

@end

@implementation IncompressibleMassBalance

   @eq cont m_dot_out (m_dot_in)
      {return m_dot_in;}

   @eq cont mass[i] (rho, vol)
      {return rho*vol;}

   @eq cont c (m_dot_in, aquer, rho)
      {return m_dot_in/aquer/rho;}

@end
```

B.2.2 „energy.smi“

Die mathematische Modellierung der Energiebilanzierungsklassen steht in Kapitel 2.3.2 ab Seite 6.

```
#include <steam.h>
#include "arraydefs.h"

@interface BalanceBaseClass : SmileObject
{
  @export //geometry
    double N [doc: "# of discrete elements"];
    double length [doc: "total length", unit: "m"];
    double aquer [doc: "cross section area", unit:"m^2"];
    double dx  [eq, doc: "length of element", unit: "m"];
    double vol [eq, doc: "volume", min: "0.0", max: "1000.0", unit: "m^3"];

  @export //massbalance
    double mass[ND] [doc: "mass", min: 0.0, max: 1.0e5, unit: "kg"];
}
@init {
  int i;
  N = (double) ND;
  length = 1.0;
  dx = length / N;
  aquer = 0.1;
  vol = aquer * dx;
  for (i=0;i<ND;i++)
      mass[i] = dx * aquer * 1000.0;
}
@end

@implementation BalanceBaseClass

  @eq cont dx (N, length){
  return length/N; }

  @eq cont vol (dx, aquer)
  { return aquer * dx ; }

@end
```

```
@interface BalanceBaseClassCM : BalanceBaseClass
{
  @export //properties
    double c [doc: "speed", min: 0.0, max: 1.0e3, unit: "m/s"];
    double rho [doc: "density", min: 0.0, max: 25.0e3, unit: "kg/m^3"];
}
@init {
  c = 0.0;
  rho = 1000.0;
}
@end

@implementation BalanceBaseClassCM

@end
```

```
@interface ConvectiveTemperatureTransportCM : BalanceBaseClassCM
{
  @export //Tin
    double T_in [doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];

  @export //Tout
    double T_out [eq, doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];

  @export //Twall
    double T[ND] [eq, doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];
    double Q_dot[ND] [doc: "heatflow", unit: "W"];

  @export //properties
    double cp [doc: "heat capacity", min: 0.0, max: 4500.0, unit: "J/kg/K"];

  @protected
    double S[ND] [eq, doc: "source term", unit: "K/s"];

}
@init {
  int i;
  for(i=0; i < ND; i++) Q_dot[i] = 0.0;
  for(i=0; i < ND; i++) T[i] = 273.15;
  T_in = T_out = 273.15;
}

@end

@implementation ConvectiveTemperatureTransportCM

@eq cont S[i:0::ND-1] (Q_dot[i], mass[i], cp)
{ return Q_dot[i]/(mass[i]*cp); }

@eq diff T[0] (T_in, T[0], c, dx, S[0])
{ return 1/dx * c * (T_in - T[0])+ S[0];}

@eq diff T[i:1::ND-1] (T[i], T[i-1], c, dx, S[i])
{ return 1/dx * c * (T[i-1] - T[i]) + S[i] ;}

@eq cont T_out (T[ND-1])
{ return T[ND-1];}
```

```
@end



@interface ConvectiveEnthalpyTransportCM : BalanceBaseClassCM
{
  @export //hin
    double h_in [doc: "enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];

  @export //hout
    double h_out [eq, doc: "enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];

  @export //Twall
    double T[ND] [doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];
    double h[ND] [eq, doc: "spec. enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];
    double Q_dot[ND] [doc: "heatflow", unit: "W"];

  @export //properties
    double cp [doc: "heat capacity", min: 0.0, max: 4500.0, unit: "J/kg/K"];

  @protected
    double S[ND] [eq, doc: "source term", unit: "J/kg/s"];

}
@init {
  int i;
  for(i=0; i < ND; i++) Q_dot[i] = 0.0;
  for(i=0; i < ND; i++) h[i] = 0.0;
  h_in = h_out = 0.0;
  cp = 1000.0;
}

@end

@implementation ConvectiveEnthalpyTransportCM

@eq cont S[i:0::ND-1] (Q_dot[i], mass[i]) {
  return Q_dot[i]/mass[i]; }

@eq diff h[0] (h_in, h[0], c, dx, S[0])
{ return 1/dx * c * (h_in - h[0]) + S[0];}

@eq diff h[i:1::ND-1] (h[i], h[i-1], c, dx, S[i])
{ return 1/dx * c * (h[i-1] - h[i])+ S[i]; }

@eq cont h_out (h[ND-1])
{ return h[ND-1]; }

@end



@interface BalanceBaseClassDM : BalanceBaseClass
{
  @export //properties
    double c[ND+1] [doc: "speed", min: 0.0, max: 1.0e3, unit: "m/s"];
    double rho[ND] [doc: "density", min: 0.0, max: 25.0e3, unit: "kg/m^3"];
}
@init {
  int i;
  for(i=0; i<ND+1; i++)
```

```
    c[i] = 0.0;
  for(i=0; i<ND; i++)
    rho[i] = 1000.0;
}
@end

@implementation BalanceBaseClassDM

@end




@interface ConvectiveTemperatureTransportDM : BalanceBaseClassDM
{
  @export //Tin
    double T_in [doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];

  @export //Tout
    double T_out [eq, doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];

  @export //Twall
    double T[ND] [eq, doc: "temperature", min: 0.0, max: 5000.0, unit: "K"];
    double Q_dot[ND] [doc: "heatflow", unit: "W"];

  @export //properties
    double cp [doc: "heat capacity", min: 0.0, max: 4500.0, unit: "J/kg/K"];

  @protected
    double S[ND] [eq, doc: "source term", unit: "K/s"];

}
@init {
  int i;
  for(i=0; i < ND; i++) Q_dot[i] = 0.0;
  for(i=0; i < ND; i++) T[i] = 273.15;
  T_in = T_out = 273.15;
  cp = 1000.0;
}

@end

@implementation ConvectiveTemperatureTransportDM

@eq cont S[i:0::ND-1] (Q_dot[i], mass[i], cp)
{ return Q_dot[i]/(mass[i]*cp); }

@eq diff T[0] (T_in, T[0], c[0], dx, S[0])
{ return 1/dx * c[0] * (T_in - T[0])+ S[0];}

@eq diff T[i:1::ND-1] (T[i], T[i-1], c[i], dx, S[i])
{ return 1/dx * c[i] * (T[i-1] - T[i]) + S[i] ;}

@eq cont T_out (T[ND-1])
{ return T[ND-1];}

@end




@interface ConvectiveEnthalpyTransportDM : BalanceBaseClassDM
{
  @export //hin
```

```
    double h_in [doc: "enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];

  @export //hout
    double h_out [eq, doc: "enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];

  @export
    double h[ND] [eq, doc: "spec. enthalpy", min: 0.0, max: 10.0e6, unit: "J/kg"];
    double Q_dot[ND] [doc: "heatflow", unit: "W"];

  @export //properties
    double cp [doc: "heat capacity", min: 0.0, max: 4500.0, unit: "J/kg/K"];

  @protected
    double S[ND] [eq, doc: "source term", unit: "J/kg/s"];

}
@init {
  int i;
  for(i=0; i < ND; i++) Q_dot[i] = 0.0;
  for(i=0; i < ND; i++) h[i] = 0.0;
  h_in = h_out = 0.0;
  cp = 1000.0;
}

@end

@implementation ConvectiveEnthalpyTransportDM

@eq cont S[i:0::ND-1] (Q_dot[i], mass[i]) {
  return Q_dot[i]/mass[i]; }

@eq diff h[0] (h_in, h[0], c[0], dx, S[0])
{ return 1/dx *c[0] * (h_in - h[0]) + S[0];}

@eq diff h[i:1::ND-1] (h[i], h[i-1], c[i], dx, S[i])
{ return 1/dx * c[i] * (h[i-1] - h[i])+ S[i]; }

@eq cont h_out (h[ND-1])
{ return h[ND-1]; }

@end
```

B.2.3 „RgEnergiebilanz.smi“

```
#include "energy.smd"
#include "arraydefs.h"
#include "consts.h"

@interface RgEnergiebilanz : ConvectiveTemperatureTransportCM{

   @export
      double aquer    [doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];
      double rho      [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double mass[ND] [eq, doc: "Masse eines diskreten Elements", unit: "kg", min: 0.0,
                       max: 1.0e+05];
      double c        [eq, doc: "Geschwindigkeit", unit: "m/s", min: 0.0, max: 1000.0];
      double m_dot    [doc: "Massenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double p        [doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double h[ND]    [eq, doc: "Enthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
   }
```

```
   @init{
         int i;
         for (i=0; i<ND; i++)
            {T[i]     = 473.15;
            }
         aquer  = 5.0;
         p      = 1.0e+05;
         m_dot  = 10.0;
         length = 1.0;
         rho    = 1.0;
         c      = 3.0;
        }

@end

@implementation RgEnergiebilanz

   @eq cont mass[j] (dx, aquer, rho)
      {return dx*aquer*rho;}

   @eq cont c (m_dot, aquer, rho)
      {return m_dot/aquer/rho;}

   @eq cont h[j] (T[j], cp)
      {return cp*T[j];}

@end
```

B.2.4 „MassenUndEnergiebilanz.smi“

```
#include "arraydefs.h"
#include "mass.smd"
#include "energy.smd"

@interface MassenUndEnergiebilanzEnthalpieCM : BalanceBaseClassCM{

   @export
      double T[ND]      [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double h[ND]      [doc: "Enthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double m_dot_in  [doc: "Eingangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double m_dot_out [doc: "Ausgangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho        [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double h_in       [doc: "Eintrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double h_out      [doc: "Austrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double cp         [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];

   @protected
      ConcentratedMassBalance*          massbil;
      ConvectiveEnthalpyTransportCM*    enerbil;
   }

@end

@implementation MassenUndEnergiebilanzEnthalpieCM

   @component massbil{}
   @component enerbil{}

   @connect{
```

```
      N               = massbil.N         = enerbil.N;
      length          = massbil.length    = enerbil.length;
      aquer           = massbil.aquer     = enerbil.aquer;
      mass[i]         = massbil.mass[i]   = enerbil.mass[i];
      c               = massbil.c         = enerbil.c;

      T[i]            = enerbil.T[i];
      h[i]            = enerbil.h[i];
      Q_dot[i]        = enerbil.Q_dot[i];
      m_dot_in        = massbil.m_dot_in;
      m_dot_out       = massbil.m_dot_out;
      rho             = massbil.rho;

      h_in            = enerbil.h_in;
      h_out           = enerbil.h_out;
      cp              = enerbil.cp;
         }

@end




@interface MassenUndEnergiebilanzEnthalpieDM : BalanceBaseClassDM{

   @export
//      double T[ND]       [doc: "Temperatur", unit: "K"];
      double h[ND]       [doc: "Enthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double Q_dot[ND]   [doc: "Waermeleistung", unit: "W"];
      double m_dot[ND+1] [doc: "Massenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho[ND]     [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double h_in        [doc: "Eintrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double h_out       [doc: "Austrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double cp          [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];

   @protected
      DistributedMassBalance*           massbil;
      ConvectiveEnthalpyTransportDM*    enerbil;
   }

@end

@implementation MassenUndEnergiebilanzEnthalpieDM

   @component massbil{}
   @component enerbil{}

   @connect{
      N               = massbil.N         = enerbil.N;
      length          = massbil.length    = enerbil.length;
      aquer           = massbil.aquer     = enerbil.aquer;
      mass[i]         = massbil.mass[i]   = enerbil.mass[i];
      c[i]            = massbil.c[i]      = enerbil.c[i];

//      T[i]            = enerbil.T[i];
      h[i]            = enerbil.h[i];
      Q_dot[i]        = enerbil.Q_dot[i];
      m_dot[i]        = massbil.m_dot[i];
      rho[i]          = massbil.rho[i];

      h_in            = enerbil.h_in;
      h_out           = enerbil.h_out;
      cp              = enerbil.cp;
         }
```

```
@end



@interface MassenUndEnergiebilanzTemperaturCM : BalanceBaseClassCM{

   @export
      double T[ND]      [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double m_dot_in  [doc: "Eingangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double m_dot_out [doc: "Ausgangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho        [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double T_in       [doc: "Eintrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double T_out      [doc: "Austrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double cp         [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];

   @protected
      ConcentratedMassBalance*             massbil;
      ConvectiveTemperatureTransportCM*    enerbil;
   }

@end

@implementation MassenUndEnergiebilanzTemperaturCM

   @component massbil{}
   @component enerbil{}

   @connect{
      N                = massbil.N          = enerbil.N;
      length           = massbil.length     = enerbil.length;
      aquer            = massbil.aquer      = enerbil.aquer;
      mass[i]          = massbil.mass[i]    = enerbil.mass[i];
      c                = massbil.c          = enerbil.c;

      T[i]             = enerbil.T[i];
      Q_dot[i]         = enerbil.Q_dot[i];
      m_dot_in         = massbil.m_dot_in;
      m_dot_out        = massbil.m_dot_out;
      rho              = massbil.rho;

      T_in             = enerbil.T_in;
      T_out            = enerbil.T_out;
      cp               = enerbil.cp;
         }

@end



@interface MassenUndEnergiebilanzTemperaturDM : BalanceBaseClassDM{

   @export
      double T[ND]        [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double Q_dot[ND]    [doc: "Waermeleistung", unit: "W"];
      double m_dot[ND+1] [doc: "Massenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho[ND]      [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double T_in         [doc: "Eintrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double T_out        [doc: "Austrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double cp           [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];
```

```
   @protected
      DistributedMassBalance*              massbil;
      ConvectiveTemperatureTransportDM*    enerbil;
   }

@end

@implementation MassenUndEnergiebilanzTemperaturDM

   @component massbil{}
   @component enerbil{}

   @connect{
      N               = massbil.N          = enerbil.N;
      length          = massbil.length     = enerbil.length;
      aquer           = massbil.aquer      = enerbil.aquer;
      mass[i]         = massbil.mass[i]    = enerbil.mass[i];
      c[i]            = massbil.c[i]       = enerbil.c[i];
      T[i]            = enerbil.T[i];
      Q_dot[i]        = enerbil.Q_dot[i];
      m_dot[i]        = massbil.m_dot[i];
      rho[i]          = massbil.rho[i];

      T_in            = enerbil.T_in;
      T_out           = enerbil.T_out;
      cp              = enerbil.cp;
         }

@end



@interface MassenUndEnergiebilanzEnthalpieInkompressibel : BalanceBaseClassCM{

   @export
      double T[ND]     [eq, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double h[ND]     [doc: "Enthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double m_dot_in  [doc: "Eingangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double m_dot_out [doc: "Ausgangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho       [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double h_in      [doc: "Eintrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double h_out     [doc: "Austrittsenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double cp        [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];

   @protected
      IncompressibleMassBalance*       massbil;
      ConvectiveEnthalpyTransportCM*   enerbil;
   }

@end

@implementation MassenUndEnergiebilanzEnthalpieInkompressibel

   @component massbil{}
   @component enerbil{}

   @connect{
      N               = massbil.N          = enerbil.N;
      length          = massbil.length     = enerbil.length;
      aquer           = massbil.aquer      = enerbil.aquer;
      mass[i]         = massbil.mass[i]    = enerbil.mass[i];
      c               = massbil.c          = enerbil.c;
```

```
      T[i]            = enerbil.T[i];
      h[i]            = enerbil.h[i];
      Q_dot[i]        = enerbil.Q_dot[i];
      m_dot_in        = massbil.m_dot_in;
      m_dot_out       = massbil.m_dot_out;
      rho             = massbil.rho;

      h_in            = enerbil.h_in;
      h_out           = enerbil.h_out;
      cp              = enerbil.cp;
         }

   @eq cont T[i] (cp, h[i])
      {return h[i]/cp + 273.15;}

@end



@interface MassenUndEnergiebilanzTemperaturInkompressibel : BalanceBaseClassCM{

   @export
      double T[ND]     [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double m_dot_in  [doc: "Eingangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double m_dot_out [doc: "Ausgangsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+06];
      double rho       [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double T_in      [doc: "Eintrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double T_out     [doc: "Austrittstemperatur", unit: "J/kg", min: 0.0, max: 5000.0];
      double cp        [doc: "spezifische Waermekapazitaet", unit: "J/kg/K"];

   @protected
      IncompressibleMassBalance*           massbil;
      ConvectiveTemperatureTransportCM*    enerbil;
   }

@end

@implementation MassenUndEnergiebilanzTemperaturInkompressibel

   @component massbil{}
   @component enerbil{}

   @connect{
      N               = massbil.N          = enerbil.N;
      length          = massbil.length     = enerbil.length;
      aquer           = massbil.aquer      = enerbil.aquer;
      mass[i]         = massbil.mass[i]    = enerbil.mass[i];
      c               = massbil.c          = enerbil.c;

      T[i]            = enerbil.T[i];
      Q_dot[i]        = enerbil.Q_dot[i];
      m_dot_in        = massbil.m_dot_in;
      m_dot_out       = massbil.m_dot_out;
      rho             = massbil.rho;

      T_in            = enerbil.T_in;
      T_out           = enerbil.T_out;
      cp              = enerbil.cp;
         }

@end
```

B.2.5 „MassenUndEnergiebilanzRohr.smi“

```
#include "consts.h"
#include "MassenUndEnergiebilanz.smd"

@interface MassenUndEnergiebilanzTemperaturRohrCM : MassenUndEnergiebilanzTemperaturCM{

   @export
      double aquer    [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
      double di       [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes   [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzTemperaturRohrCM

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end



@interface MassenUndEnergiebilanzTemperaturRohrDM : MassenUndEnergiebilanzTemperaturDM{

   @export
      double aquer    [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
      double di       [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes   [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzTemperaturRohrDM

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end
```

```
@interface MassenUndEnergiebilanzEnthalpieRohrCM : MassenUndEnergiebilanzEnthalpieCM{

   @export
      double aquer     [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
      double di        [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes    [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzEnthalpieRohrCM

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end




@interface MassenUndEnergiebilanzEnthalpieRohrDM : MassenUndEnergiebilanzEnthalpieDM{

   @export
      double aquer     [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
      double di        [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes    [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzEnthalpieRohrDM

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end




@interface MassenUndEnergiebilanzTemperaturInkompressibelRohr :
             MassenUndEnergiebilanzTemperaturInkompressibel{

   @export
      double aquer     [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
```

```
      double di       [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes   [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzTemperaturInkompressibelRohr

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end




@interface MassenUndEnergiebilanzEnthalpieInkompressibelRohr :
             MassenUndEnergiebilanzEnthalpieInkompressibel{

   @export
      double aquer    [eq, doc: "Gesamtquerschnittsflaeche", unit: "m^2", min: 0.0];

   @protected
      double di       [doc: "Rohrinnendurchmesser", unit: "m"];
      double ntubes   [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
         aquer  = 1.0;
         di     = 20.0e-03;
         ntubes = 1.0;
        }

@end

@implementation MassenUndEnergiebilanzEnthalpieInkompressibelRohr

   @eq cont aquer (di, ntubes)
      {return ntubes*PI/4.0*di*di;}

@end
```

B.2.6 „Impuls.smi“

Die mathematische Modellierung der Impulsbilanzierungsklassen steht in Kapitel 2.3.3 ab Seite 8.

```
#include "energy.smd"
#include "arraydefs.h"
#include "consts.h"

#define TAU 0.01
```

```
@interface ImpulsbilanzStationaer : BalanceBaseClassDM{

   @export
      double c[ND+1]     [eq, doc: "Geschwindigkeit", unit: "m/s", min: 0.0, max: 1000.0];
      double m_dot_in    [doc: "Eintrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double m_dot_out   [doc: "Austrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double m_dot[ND+1] [eq, doc: "Massenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double p[ND]       [doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double rho_in      [doc: "Eintrittsdichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double di          [doc: "Rohrinnendurchmesser", unit: "m", min: 0.0];

   @protected
      double lambda_r    [doc: "Druckverlustbeiwert", unit: "1", min: 0.0];
      }

   @init{
        }
@end

@implementation ImpulsbilanzStationaer

   @eq cont m_dot[0] (m_dot_in)
      {return m_dot_in;}

   @eq cont m_dot[i:1::ND-1] (rho[i], rho[i-1], c[i], aquer)
      {return 0.5*(rho[i]+rho[i-1])*c[i]*aquer;}

   @eq cont m_dot[ND] (m_dot_out)
      {return m_dot_out;}

   @eq cont c[0] (m_dot_in, aquer, rho_in)
      {return m_dot_in/aquer/rho_in;}

   @eq cont c[i:1::ND-1] (p[i], p[i-1], rho[i], rho[i-1], di, dx, lambda_r)
      {
         double y;  // Hilfsvariable

         y = 4.0 * (p[i-1]-p[i])*di/(dx*lambda_r*(rho[i-1]+rho[i]));
         if      (y > 1.0)  {return sqrt (y);}
         else if (y < -1.0) {return -sqrt (-y);}
         else                {
                              return (-0.25 * y * y + 1.25) * y;
                             }
      }

   @eq cont c[ND] (m_dot_out, aquer, rho[ND-1])
      {return m_dot_out/aquer/rho[ND-1];}

@end
```

```
@interface ImpulsbilanzInstationaer : BalanceBaseClassDM{

   @export
      double c[ND+1]     [eq, doc: "Geschwindigkeit", unit: "m/s", min: 0.0, max: 1000.0];
      double m_dot_in    [doc: "Eintrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double m_dot_out   [doc: "Austrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double m_dot[ND+1] [eq, doc: "Massenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double p[ND]       [doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double rho_in      [doc: "Eintrittsdichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double di          [doc: "Rohrinnendurchmesser", unit: "m", min: 0.0];
```

```
   @protected
      double lambda_r    [doc: "Druckverlustbeiwert", unit: "1", min: 0.0];
      double phi[ND]     [doc: "Zweiphasenmultiplikator", unit: "1"];
      }

   @init{
         int i;
         for (i=0; i<ND; i++)
            {phi[i] = 1.0;}           // Einphasengebiet
        }
@end

@implementation ImpulsbilanzInstationaer

   @eq diff m_dot[0] (m_dot_in, m_dot[0])
      {return 1/TAU*(m_dot_in-m_dot[0]);}

   @eq diff m_dot[i:1::ND-1] (aquer, p[i-1], p[i], dx, lambda_r,
                              di, m_dot[i], rho[i-1], rho[i], phi[i])
      {return aquer*((p[i-1]-p[i])/dx-phi[i]*lambda_r/di/aquer/aquer*
       m_dot[i]*fabs(m_dot[i])/(rho[i-1]+rho[i]));}

   @eq diff m_dot[ND] (m_dot_out, m_dot[ND])
      {return 1/TAU*(m_dot_out-m_dot[ND]);}

   @eq cont c[0] (m_dot[0], aquer, rho_in)
      {return m_dot[0]/aquer/rho_in;}

   @eq cont c[i:1::ND-1] (m_dot[i], aquer, rho[i-1])
      {return m_dot[i]/aquer/rho[i-1];}

   @eq cont c[ND] (m_dot[ND], aquer, rho[ND-1])
      {return m_dot[ND]/aquer/rho[ND-1];}

@end



@interface ImpulsbilanzInstationaer2Ph : ImpulsbilanzInstationaer{

   @export
      double x[ND]            [doc: "Dampfgehalt", unit: "1", min: 0.0, max: 1.0];

   @protected
      double phi[ND]     [eq, doc: "Zweiphasenmultiplikator", unit: "1"];
      double F[ND]       [eq, doc: "Druckfunktion", unit: "1"];
      double p_stern[ND] [eq, doc: "bezogener Siededruck", unit: "Pa", min: 0.0,
                          max: 1000.0e+05];
      }

   @init{
        }

@end

@implementation ImpulsbilanzInstationaer2Ph

   @eq cont p_stern[i] (p[i])
      {return p[i]/PKRIT;}

   @eq cont F[i] (p_stern[i])
      {return 4.75+p_stern[i]*(9.03+p_stern[i]*(-21.25+p_stern[i]*24.01));}
```

```
   @eq cont phi[i] (p_stern[i], F[i], x[i])
      {return 1+(1-p_stern[i])/p_stern[i]*F[i]*x[i]*(1+1.5*pow(1-x[i], 0.36));}

@end



@interface ImpulsbilanzInkompressibel : BalanceBaseClassCM{

   @export
      double p_in       [eq, doc: "Eintrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double p_out      [doc: "Austrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double di          [doc: "Rohrinnendurchmesser", unit: "m", min: 0.0];

   @protected
      double lambda_r   [doc: "Druckverlustbeiwert", min: 0.0, unit: "1"];
   }

   @init{
         p_in  = 1.0e+05;
         p_out = 1.0e+05;
        }

@end

@implementation ImpulsbilanzInkompressibel

   @eq cont p_in (p_out, lambda_r, length, di, rho, c)
      {return p_out + lambda_r * length / di * rho * 0.5 * c * c;}

@end
```

B.2.7 „Wasserbilanz.smi"

```
#include "MassenUndEnergiebilanzRohr.smd"
#include "Impuls.smd"
#include "SWFDampf.smd"
#include "Druckspeicher.smd"
#include "arraydefs.h"
#include <steam.h>

#define TAU 0.01

/* Daten fuer Zwangdurchlauf-Abhitzekessel Stuttgart (Winterbetrieb) */

@interface WasserbilanzCM_cuts : BalanceBaseClassCM{

   @export
      double T[ND]      [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double h_in       [doc: "spezif. Eintrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double p_in       [doc: "Eintrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double rho_in     [doc: "Eintrittsdichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double m_dot_in  [doc: "Eintrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double h_out      [doc: "spezif. Austrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double p_out      [doc: "Austrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double m_dot_out [doc: "Austrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
}
```

```
@end

@implementation WasserbilanzCM_cuts

@end




@interface WasserbilanzDM_cuts : BalanceBaseClassDM{

   @export
      double T[ND]      [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double Q_dot[ND] [doc: "Waermeleistung", unit: "W"];
      double h_in      [doc: "spezif. Eintrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double p_in      [doc: "Eintrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double rho_in    [doc: "Eintrittsdichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double m_dot_in  [doc: "Eintrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
      double h_out     [doc: "spezif. Austrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double p_out     [doc: "Austrittsdruck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double m_dot_out [doc: "Austrittsmassenstrom", unit: "kg/s", min: 0.0, max: 1.0e+05];
}

@end

@implementation WasserbilanzDM_cuts

@end




@interface WasserbilanzEcorohr : WasserbilanzCM_cuts{

   @protected
      MassenUndEnergiebilanzEnthalpieInkompressibelRohr*   mue;
      PressureStore*                                       speicher;
      ImpulsbilanzInkompressibel*                          impuls;
   }

   @init{
        }

@end

@implementation WasserbilanzEcorohr

   @component mue{
                int i;
                for (i=0; i<ND; i++)
                   {h[i] = 442.99e+03;}
                di     = 19.8e-03;
                ntubes = 32;
                cp     = 4219.0;
               }

   @component speicher{
                     elastic = 3.734e+06;   /* -> p = 41.7 bar */
                     m       = 100.0;
                     V_0     = 1.0;
                    }
```

```
   @component impuls{
                     lambda_r  = 0.02;
                     di        = 19.8e-03;
                    }

   @connect{
      length        = mue.length         = impuls.length   = 72.8; /* = 4.550*16 */
      aquer         = mue.aquer          = impuls.aquer    = 0.00985304;
      mass[i]       = mue.mass[i]        = 114.33; /* vol=length/ND*aquer=0.11955022 */
      Q_dot[i]      = mue.Q_dot[i];
      T[i]          = mue.T[i]           = 378.15; /* = 105.00 grdC */
      c             = mue.c              = impuls.c        = 0.27;
      h_in          = mue.h_in           = 442.99e+03;
      h_out         = mue.h_out          = 1097.88e+03;
      m_dot_in      = mue.m_dot_in       = 2.5116;
      mue.m_dot_out = speicher.m_dot_1   = 2.5116;
      m_dot_out     = speicher.m_dot_2   = 2.5116;
      rho = rho_in = speicher.rho_1 = speicher.rho_2 = mue.rho = impuls.rho = 956.31;
      p_in          = impuls.p_in        = 41.7e+05;
      p_out = speicher.p = speicher.p_1 = speicher.p_2 = impuls.p_out = 41.7e+05;
   }

@end




@interface WasserbilanzVerdampferrohr : WasserbilanzDM_cuts{

   @export
      double di                     [doc: "Rohrinnendurchmesser", unit: "m"];

   @protected
      MassenUndEnergiebilanzEnthalpieRohrDM*   mue;
      ImpulsbilanzInstationaer*                impuls;
      SWFNassdampfPUndTAusHUndRho*             swf;
      }

   @init{
         m_dot_in       = 2.5116;
         m_dot_out      = 2.5116;
        }
@end

@implementation WasserbilanzVerdampferrohr

   @component mue{
                     ntubes = 32;
                     di     = 19.8e-03;
                 }

   @component impuls{
                     lambda_r       = 0.02;
                    }

   @component swf{
         int i;
         for (i=0; i<ND; i++)
            {
             x[i]     = 0.0;
             r[i]     = 1697.8e+03;
             rho_1[i] = 794.34;
             rho_2[i] = 21.14;
```

```
            h_1[i]   = 1101.6e+03;
            h_2[i]   = 2799.4e+03;
            nassdampfcheck[i] = 1.0;
           }
       }

   @connect{
      length    = mue.length  = impuls.length  = 86.45;
      aquer     = mue.aquer   = impuls.aquer   = 0.0098530;
      mass[i]   = mue.mass[i] = impuls.mass[i] = 112.68; /* vol=length/ND*aquer=0.14196531 */
      di        = impuls.di                                          = 19.8e-03;
      Q_dot[i]  = mue.Q_dot[i]                                      = 0.0;
      T[i]      = swf.T[i];
      c[i]      = mue.c[i]   = impuls.c[i]                  = 0.32; /* Eintritt */
      rho[i]    = mue.rho[i] = impuls.rho[i] = swf.rho[i] = 793.7; /* Eintritt */
      rho_in    = impuls.rho_in                                     = 793.7;
      h_in      = mue.h_in                                          = 1097.88e+03;
      h_out     = mue.h_out                                         = 2805.00e+03;
      m_dot_in  = impuls.m_dot_in;
      m_dot_out = impuls.m_dot_out;
      p_out     = impuls.p[ND-1]      = swf.p[ND-1];

                 mue.h[i]            = swf.h[i]         = 1097.88e+03;
                 mue.m_dot[i]        = impuls.m_dot[i]  = 2.5116;
      p_in      = impuls.p[0]         = swf.p[0]          = 41.0e+05;
                 impuls.p[1]         = swf.p[1]         = 40.9e+05;
                 impuls.p[2]         = swf.p[2]         = 40.8e+05;
                 impuls.p[3]         = swf.p[3]         = 40.7e+05;
                 impuls.p[4]         = swf.p[4]         = 40.6e+05;
                 impuls.p[5]         = swf.p[5]         = 40.5e+05;
   }

@end




@interface WasserbilanzUeberhitzerrohr : WasserbilanzDM_cuts{

   @export
      double di                     [doc: "Rohrinnendurchmesser", unit: "m"];

   @protected
      MassenUndEnergiebilanzEnthalpieRohrDM*   mue;
      ImpulsbilanzInstationaer*                impuls;
      SWFUebhDampfPUndTAusHUndRho*             swf;
//       SWFUebhDampfPUndTAusHUndRhoIdGas*        swf;
   }

   @init{
         m_dot_in  = 2.5116;
         m_dot_out = 2.5116;
        }

@end

@implementation WasserbilanzUeberhitzerrohr

   @component mue{
                    ntubes = 32;
                    di     = 19.8e-03;
                 }

   @component impuls{
```

```
                  lambda_r       = 0.02;
                 }

   @component swf{}

   @connect{
      length    = mue.length         = impuls.length  = 27.180;    /* = 4.530*6 */
      aquer     = mue.aquer          = impuls.aquer   = 0.0098530;
      mass[i]   = mue.mass[i]        = impuls.mass[i] = 0.9150;
      di        = impuls.di                           = 19.8e-03;
      Q_dot[i]  = mue.Q_dot[i];
      T[i]      = swf.T[i]                            = 524.6; /* = 251.48 grdC */
   // Hiermit wird auch T[i][|hist|] gesetzt
      c[i]      = mue.c[i]           = impuls.c[i]      = 12.44;
      rho[0]    = swf.rho[0]         = impuls.rho[0]    = mue.rho[0] = 20.40;
      rho[1]    = swf.rho[1]         = impuls.rho[1]    = mue.rho[1] = 20.30;
      rho[2]    = swf.rho[2]         = impuls.rho[2]    = mue.rho[2] = 20.20;
      rho[3]    = swf.rho[3]         = impuls.rho[3]    = mue.rho[3] = 20.10;
      rho[4]    = swf.rho[4]         = impuls.rho[4]    = mue.rho[4] = 20.00;
      rho[5]    = swf.rho[5]         = impuls.rho[5]    = mue.rho[5] = 19.90;
      rho_in    = impuls.rho_in                                      = 19.90;
      h_in      = mue.h_in                                           = 2805.0e+03;
      h_out     = mue.h_out                                          = 3419.1e+03;
      m_dot_in  = impuls.m_dot_in;
      m_dot_out = impuls.m_dot_out;
      p_out     = impuls.p[ND-1]     = swf.p[ND-1];

                  mue.h[i]           = swf.h[i]         = 2805.0e+03;
                  mue.m_dot[i]       = impuls.m_dot[i]  = 2.5116;
      p_in      = impuls.p[0]        = swf.p[0]         = 40.782e+05;
                  impuls.p[1]        = swf.p[1]         = 40.682e+05;
                  impuls.p[2]        = swf.p[2]         = 40.582e+05;
                  impuls.p[3]        = swf.p[3]         = 40.482e+05;
                  impuls.p[4]        = swf.p[4]         = 40.382e+05;
                  impuls.p[5]        = swf.p[5]         = 40.282e+05;
   }

@end
```

B.3 Wärmeübertragung

B.3.1 „Wue.smi“

Die mathematische Modellierung der Wärmeübergangsklassen ist in Kapitel 2.4.2 ab Seite 13 zu finden, die der Wärmestrahlungsklassen in Kapitel 2.4.3 ab Seite 16.

```
#include "arraydefs.h"
#include "consts.h"

@interface Wue : SmileObject{

        @export
           double dQ[ND] [doc: "Waermeleistung", unit: "W"];
           double A      [doc: "Gesamtflaeche", unit: "m^2"];

        @protected
           double n      [doc: "Anzahl diskreter Elemente", unit: "1"];
           double f      [doc: "Wandflaechenfaktor", unit: "1"];
```

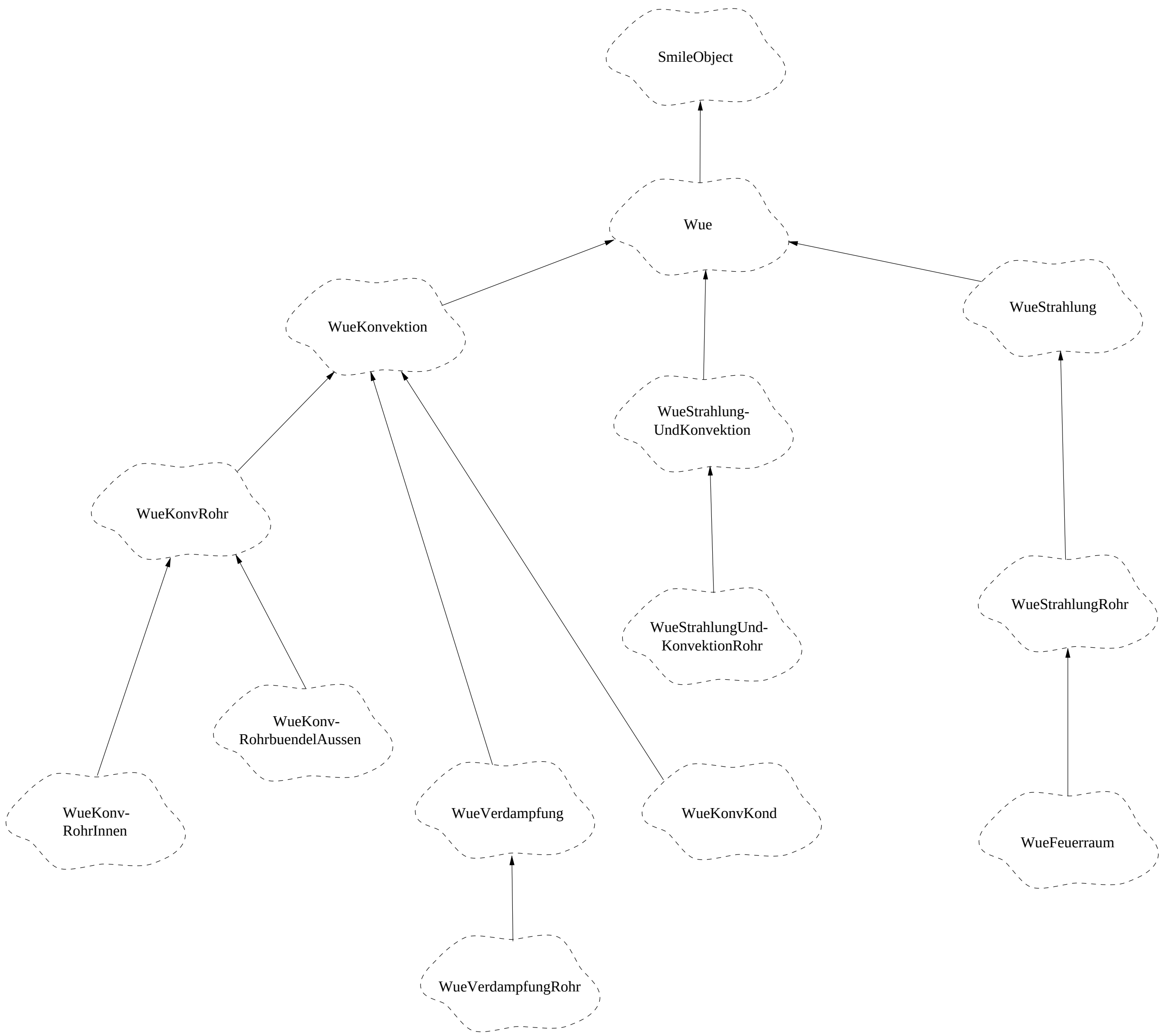

Abbildung 21: Klassenstruktur Wärmeübergang und -strahlung

Klassenname	Elternklasse	`@export`-Variable	Gleichung
Wue	SmileObject	`dQ[ND]`	
		`A`	
WueKonvektion	Wue	`dQ[ND]`	×
		`t_fluid[ND]`	
		`t_wall[ND]`	
		`c`	
WueKonvektion_D	Wue	`dQ[ND]`	×
		`t_fluid[ND]`	
		`t_wall[ND]`	
		`c`	
WueKonvRohr	WueKonvektion		
WueKonvRohrInnen	WueKonvRohr	`A`	×
WueKonvRohrbuendelAussen	WueKonvRohr	`A`	×
WueStrahlung	Wue	`dQ[ND]`	×
		`t_strlg[ND]`	
		`t_wall[ND]`	
WueStrahlungUndKonvektion	Wue	`dQ_strlg[ND]`	×
		`dQ_konv[ND]`	×
		`dQ[ND]`	×
		`t_fluid[ND]`	
		`t_wall[ND]`	
WueStrahlungRohr	WueStrahlung	`A`	×
WueStrahlungUndKonvektionRohr	WueStrahlungUndKonvektion	`A`	×
WueFeuerraum	WueStrahlungRohr	`m_dot_br`	
		`m_dot_l`	
		`m_dot_rg`	×
		`t_FA`	×
		`t_strlg[ND]`	×
WueVerdampfung	WueKonvektion_D		
WueVerdampfungRohr	WueVerdampfung	`A`	×
WueKonvKond	WueKonvektion		

Tabelle 11: `@export`-Variablen der Wärmeübergangs- und -strahlungsklassen

```
/*                              beruecksichtigt Wandflaechenveraenderung,
                              z. B. durch einseitige Beaufschlagung oder
                              durch Flossen oder Rippen */
          }

        @init{
          int i;
          for (i=0; i<ND; i++)
             dQ[i] = 0.0;
          A = 1.0;
          n = ND-1;
          f = 1.0;
          }
@end

@implementation Wue

@end




@interface WueKonvektion : Wue{

        @export
           double dQ[ND]       [eq, doc: "Waermeleistung", unit: "W"];
           double t_fluid[ND] [doc: "Fluidtemperatur", unit: "K", min: 0.0, max: 5000.0];
           double t_wall[ND]  [doc: "Wandtemperatur", unit: "K", min: 0.0, max: 5000.0];
           double c            [doc: "Fluidgeschwindigkeit", unit: "m/s", min: 0.0,
                                max: 1000.0];

        @protected
           double alpha        [doc: "Waermeuebergangskoeffizient", min: 0.0,
                                unit: "W/m^2/K"];
           double Nu           [doc: "Nusselt-Zahl", unit: "1"];
                }

        @init{
           int i;
           for (i=0; i<ND; i++)
              {
              dQ[i]      = 0.0;
              t_fluid[i] = 300.0;
              t_wall[i]  = 400.0;
              }
           c       = 1.0;
           alpha   = 1.0e+03;
           Nu      = 100.0;
           }

@end

@implementation WueKonvektion

        @eq cont dQ[i] (alpha, A, t_wall[i], t_fluid[i], n, f)
           {return alpha*f*A/n*(t_wall[i]-t_fluid[i]);}

@end



@interface WueKonvektion_D : Wue{           /* Distributed alphas */
```

```
        @export
           double dQ[ND]      [eq, doc: "Waermeleistung", unit: "W"];
           double t_fluid[ND] [doc: "Fluidtemperatur", unit: "K", min: 0.0, max: 5000.0];
           double t_wall[ND]  [doc: "Wandtemperatur", unit: "K", min: 0.0, max: 5000.0];
           double c           [doc: "Fluidgeschwindigkeit", unit: "m/s", min: 0.0, max: 1000.0];

        @protected
           double alpha[ND]   [doc: "Waermeuebergangskoeffizient", min: 0.0, unit: "W/m^2/K"];
           double Nu          [doc: "Nusselt-Zahl", unit: "1"];
                }

        @init{
           int i;
           for (i=0; i<ND; i++)
              {
              dQ[i]      = 0.0;
              t_fluid[i] = 300.0;
              t_wall[i]  = 400.0;
              alpha[i]   = 1.0e+03;
              }
           c       = 1.0;
           Nu      = 100.0;
           }

@end

@implementation WueKonvektion_D

        @eq cont dQ[i] (alpha, A, t_wall[i], t_fluid[i], n, f)
           {return alpha[i]*f*A/n*(t_wall[i]-t_fluid[i]);}

@end



@interface WueKonvRohr : WueKonvektion{

         @protected
           double alpha     [eq, doc: "Waermeuebergangskoeffizient", min: 0.0, unit: "W/m^2/K"];
           double lambda    [doc: "Waermeleitkoeffizient", unit: "W/m/K"];
           double Re        [eq, doc: "Reynolds-Zahl", unit: "1", min: 0.0];
           double Nu        [doc: "Nusselt-Zahl", unit: "1"];
           double l         [doc: "Rohrlaenge", unit: "m"];
           double di        [doc: "Innendurchmesser", unit: "m"];
           double da        [doc: "Aussendurchmesser", unit: "m"];
           double L_0       [doc: "Charakteristische Laenge", unit: "m"];
           double eta       [doc: "Dynamische Viskositaet", unit: "Ns/m^2"];
           double rho       [doc: "Dichte", unit: "kg/m^3"];
           double Pr        [doc: "Prandtl-Zahl", unit: "1"];
           double n_tubes   [doc: "Anzahl der Rohre", unit: "1"];
           }

        @init{
           alpha   = 1.0e+03;
           lambda  = 0.6;
           Re      = 8000.0;
           Nu      = 100.0;
           l       = 1.0;
           di      = 20.0e-03;
           da      = 24.0e-03;
           L_0     = 1.0;
           eta     = 1.0e-03;
           rho     = 1.0e+03;
```

```
            Pr      = 7.0;
            n_tubes = 1.0;
            }

@end

@implementation WueKonvRohr

        @eq cont Re (c, L_0, eta, rho)
           {return fabs(c)*L_0*rho/eta;}

        @eq cont alpha (Nu, lambda, L_0)
           {return Nu*lambda/L_0;}

@end




@interface WueKonvRohrInnen : WueKonvRohr{

        @export
           double A     [eq, doc: "Rohrinnenflaeche", unit: "m^2", min: 0.0];

        @protected
           double L_0   [eq, doc: "Charakteristische Laenge", unit: "m", min: 0.0];
           double Nu    [eq, doc: "Nusselt-Zahl", unit: "1", min: 0.0];
           double xi    [doc: "Druckverlustbeiwert", unit: "1"];
           double where [eq, doc: "laminar oder turbulent"];
           }

        @init{
           L_0   = 20.0e-03;
           Nu    = 100.0;
           xi    = 0.034;
           where = 2;
           }

@end

@implementation WueKonvRohrInnen

        @eq cont A (n_tubes, di, l)
           {return n_tubes*PI*di*l;}

        @eq cont L_0 (di)
           {return di;}

        @eq discrete where (time, Re)
           {if (Re<2300.0)
               return 0;
            else if (Re<10000.0)
               return 1;
            else
               return 2;}

//        @eq cont xi   (Re)
//           {return pow(0.79*log(Re)-1.64, -2);}

        @eq cont Nu (where, Re, Pr, l, di, xi)
           {if (where == 0)
               return 4.364;
            else if (where == 1)
               return (Re-1000.0)*Pr*xi/8/(1+12.7*sqrt(xi/8)*
```

```
                    (pow(Pr, 0.66667)-1))*(1+pow(di/l, 0.66667))*
                    (Re-2300)/7700+4.364*(10000-Re)/7700;
          else
             return (Re-1000.0)*Pr*xi/8/(1+12.7*sqrt(xi/8)*
                    (pow(Pr, 0.66667)-1))*(1+pow(di/l, 0.66667));
          }

@end




@interface WueKonvRohrbuendelAussen : WueKonvRohr{

        @export
           double A       [eq, doc: "Rohraussenflaeche", unit: "m^2", min: 0.0];

        @protected
           double L_0     [eq, doc: "Charakteristische Laenge", unit: "m", min: 0.0];
           double Nu      [eq, doc: "Nusselt-Zahl", unit: "1", min: 0.0];
           double Nu_lam  [eq, doc: "Laminare Nusselt-Zahl", unit: "1", min: 0.0];
           double Nu_turb [eq, doc: "Turbulent Nusselt-Zahl", unit: "1", min: 0.0];
           double psi     [doc: "Freiraumanteil", unit: "1"];
           double Re_psi  [eq, doc: "Reynolds-Zahl", unit: "1", min: 0.0];
           double fa      [doc: "Rohrformfaktor", unit: "1"];
           }

        @init{
           L_0     = 0.1;
           Nu      = 500.0;
           Nu_lam  = 100.0;
           Nu_turb = 1000.0;
           psi     = 0.9;
           Re_psi  = 8000.0;
           fa      = 1.5;
           }

@end

@implementation WueKonvRohrbuendelAussen

        @eq cont A (n_tubes, da, l)
           {return n_tubes*PI*da*l;}

        @eq cont L_0 (da)
           {return PI/2*da;}

        @eq cont Re_psi (Re, psi)
           {return Re/psi;}

        @eq cont Nu_lam (Re_psi, Pr)
           {return 0.664*sqrt(Re_psi)*pow(Pr, 0.33333);}

        @eq cont Nu_turb (Re_psi, Pr)
           {return 0.037*pow(Re_psi, 0.8)*Pr/
                  (1+2.443*pow(Re_psi, -0.1)*(pow(Pr, 0.66667)-1));}

        @eq cont Nu (fa, Nu_lam, Nu_turb)
           {return fa*(0.3+sqrt(Nu_lam*Nu_lam+Nu_turb*Nu_turb));}

@end
```

```
@interface WueStrahlung : Wue{

        @export
           double dQ[ND]      [eq, doc: "Waermeleistung", unit: "W"];
           double t_strlg[ND] [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
           double t_wall[ND]  [doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];

        @protected
           double C_S     [doc: "Strahlungskonstante", unit: "W/m^2/K^4"];
           double epsilon [doc: "Emissionsverhaeltnis", unit: "1", min: 0.0, max: 1.0];
           }

        @init{
           int i;
           for (i=0; i<ND; i++)
              {
              dQ[i]      = 0.0;
              t_strlg[i] = 400.0;
              t_wall[i]  = 300.0;
              }
           C_S     = 5.67e-08;
           epsilon = 0.80;
           }

@end

@implementation WueStrahlung

        @eq cont dQ[i] (epsilon, C_S, A, t_strlg[i], t_wall[i], n, f)
           {return epsilon*C_S*f*A/n*(pow(t_wall[i],4)-pow(t_strlg[i],4));}

@end




@interface WueStrahlungUndKonvektion : Wue{

/* Sonderfaelle: alpha=0   -> Waermeuebertragung nur durch Strahlung
                 epsilon=0 -> Waermeuebertragung nur durch Konvektion */

        @export
           double dQ_strlg[ND]  [eq, doc: "Waermeleistung durch Strahlung", unit: "W"];
           double dQ_konv[ND]   [eq, doc: "Waermeleistung durch Konvektion", unit: "W"];
           double dQ[ND]        [eq, doc: "Gesamtwaermeleistung", unit: "W"];
           double t_fluid[ND]   [doc: "Fluidtemperatur", unit: "K", min: 0.0, max: 5000.0];
           double t_wall[ND]    [doc: "Wandtemperatur", unit: "K", min: 0.0, max: 5000.0];

        @protected
           double alpha         [doc: "Waermeuebergangskoeffizient", unit: "W/m^2/K"];
           double epsilon       [doc: "Emissionsverhaeltnis", unit: "1"];
           double C_S           [doc: "Strahlungskonstante", unit: "W/m^2/K^4"];
        }

        @init{
           int i;
           for (i=0; i<ND; i++)
              {
               dQ_strlg[i]  = 0.0;
               dQ_konv[i]   = 0.0;
               t_fluid[i]   = 400.0;
               t_wall[i]    = 300.0;
              }
```

```
          alpha   = 50.0;
          epsilon = 0.80;
          C_S     = 5.67e-08;
            }

@end

@implementation WueStrahlungUndKonvektion

        @eq cont dQ_strlg[i] (epsilon, C_S, A, t_fluid[i], t_wall[i], n, f)
           {return epsilon*C_S*f*A/n*(pow(t_wall[i],4)-pow(t_fluid[i],4));}

        @eq cont dQ_konv[i] (alpha, A, t_wall[i], t_fluid[i], n, f)
           {return alpha*f*A/n*(t_wall[i]-t_fluid[i]);}

        @eq cont dQ[i] (dQ_strlg[i], dQ_konv[i])
           {return dQ_strlg[i]+dQ_konv[i];}

@end



@interface WueStrahlungRohr : WueStrahlung{

   @export
      double A      [eq, doc: "Rohraussenflaeche", unit: "m", min: 0.0];

   @protected
      double n_tubes[doc: "Anzahl der Rohre", unit: "1"];
      double l      [doc: "Rohrlaenge", unit: "m"];
      double da     [doc: "Rohraussendurchmesser", unit: "m"];
   }

   @init{
      da         = 24.0e-03;
      l          = 1.0;
      n_tubes    = 1.0;
      }

@end

@implementation WueStrahlungRohr

      @eq cont A (da, l, n_tubes)
      {return n_tubes*PI*da*l;}

@end



@interface WueStrahlungUndKonvektionRohr : WueStrahlungUndKonvektion{

   @export
      double A      [eq, doc: "Rohraussenflaeche", unit: "m", min: 0.0];

   @protected
      double n_tubes[doc: "Anzahl der Rohre", unit: "1"];
      double l      [doc: "Rohrlaenge", unit: "m"];
      double da     [doc: "Rohraussendurchmesser", unit: "m"];
   }

   @init{
```

```
      da         = 24.0e-03;
      l          = 1.0;
      n_tubes    = 1.0;
      }

@end

@implementation WueStrahlungUndKonvektionRohr

      @eq cont A (da, l, n_tubes)
      {return n_tubes*PI*da*l;}

@end




@interface WueVerdampfung : WueKonvektion_D{

        @protected
           double alpha[ND] [eq, doc: "Waermeuebergangskoeffizient", min: 0.0, unit: "W/m^2/K"];
           double p         [doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
           }

        @init{
           int i;
           for (i=0; i<ND; i++)
              {
              dQ[i]      = 0.0;
              t_wall[i]  = 500.0;
              t_fluid[i] = 400.0;
              alpha[i]   = 3000.0;
              }
           p      = 100.0e+05;
           }

@end

@implementation WueVerdampfung

        @eq cont alpha[i] (t_wall[i], t_fluid[i], p)      /* Blasensieden */
           {return 10.86*pow(fabs(t_wall[i]-t_fluid[i]), 2.57)*pow(p/(1.0e+05), 0.86);}

@end




@interface WueVerdampfungRohr : WueVerdampfung{

   @export
      double A              [eq, doc: "Rohroberflaeche", unit: "m^2", min: 0.0];

   @protected
      double di             [doc: "Innendurchmesser", unit: "m"];
      double l              [doc: "Rohrlaenge", unit: "l"];
      double n_tubes       [doc: "Anzahl der Rohre", unit: "1"];
      }

   @init{
        A       = 0.063;
        di      = 20.0e-03;
        l       = 1.0;
        n_tubes = 1.0;
```

```
         }

@end

@implementation WueVerdampfungRohr

   @eq cont A (n_tubes, di, l)
      {return n_tubes*PI*di*l;}

@end



@interface WueKonvKond : WueKonvektion{

    @protected
       double alpha [doc: "Waermeuebergangskoeffizient", unit: "W/m^2/K"];
       }

    @init
       {
        alpha = 8000.0;
       }

@end

@implementation WueKonvKond

@end
```

B.3.2 „Feuerraum.smi“

Die mathematische Modellierung der Klasse WueFeuerraum ist in Kapitel 4.1.4 ab Seite 27 zu finden.

```
#include "Wue.smd"
#include "arraydefs.h"
#include "consts.h"

@interface WueFeuerraum : WueStrahlungRohr{

        /*smile-doc
        - description   : Waermeuebergang im Feuerraum nach Konakow-Modell
        - type          :
        - assumptions   : Verbrennungsprozess unendlich schnell,
                          Waermeuebertragung nur durch Strahlung,
                          konstante Strahlungstemperatur ueber Verdampferrohrlaenge,
                          Wand- gegenueber Strahlungstemperatur vernachlaessigbar klein
        - limitations   :
        - reference     : Gl'gen aus {KonakowP55}; {KlugM84}, S.27ff oder {DolezalR90}, S.173ff
        - maturity      :
        - numerical tips:
        - author        : Oliver Czygan <Czygan@tu-harburg.d400.de>
        - maintainer    :
        - creation date : Feb 05 1996
        - last modified : $_mtime$
        - version       : $_self$
        smile-doc*/
```

```
@export
   double m_dot_br      [doc: "Brennstoffmassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
   double m_dot_l       [doc: "Luftmassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
   double m_dot_rg      [eq, doc: "Rauchgasmassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
   double t_FA          [eq, doc: "Feuerraumaustrittstemperatur", unit: "K",
                         min: 0.0, max: 5000.0];
   double t_strlg[ND]   [eq, doc: "Strahlungstemperatur", unit: "K", min: 0.0,
                         max: 5000.0];

@protected
   double t_ad          [eq, doc: "adiabate Verbrennungstemperatur", unit: "K",
                         min: 0.0, max: 5000.0];
   double t_g           [eq, doc: "Strahlungstemperatur der Gasstrahlung", unit: "K",
                         min: 0.0, max: 5000.0];
   double cp_rg         [doc: "spezifische Waermekapazitaet Rauchgas", unit: "K/kg/K"];
   double H_u           [doc: "Heizwert", unit: "J/kg"];
   double h_b           [doc: "spezifische Brennstoffenthalpie", unit: "J/kg"];
   double cp_l          [doc: "spezifische Waermekapazitaet Luft", unit: "K/kg/K"];
   double t_l           [doc: "Lufttemperatur", unit: "K", min: 0.0, max: 5000.0];
   double a             [eq, doc: "Temperaturverhaeltnis", min: 0.0];
   double z             [doc: "Schwaerzefaktor"];
   double Ko            [eq, doc: "Konakow-Zahl", min: 0.0];
   }

@init {
   int i;
   m_dot_br   = 15.0;
   m_dot_l    = 150.0;
   m_dot_rg   = 165.0;
   t_FA       = 1000.0;
   for (i=0; i<ND; i++)
      {
      t_strlg[i] = 1600.0;
      t_wall[i]  = 600.0;
      dQ[i]      = -1.0e+04;
      }

   t_ad       = 2200.0;
   t_g        = 1600.0;    /* Trockenfeuerung Steinkohle */
   cp_rg      = 1050.0;
   H_u        = 24.0e+06;  /* Steinkohle */
   h_b        = 0.0;
   cp_l       = 1000.0;
   t_l        = 300.0;
   a          = 1.0;
   z          = 0.933;     /* Steinkohlestaubfeuerung */
   Ko         = 4000.0;
   }

@end

@implementation WueFeuerraum

   @eq cont t_strlg[i] (t_g)
      {return t_g;}

   @eq cont m_dot_rg (m_dot_br, m_dot_l)
      {return m_dot_br+m_dot_l;}

   @eq cont t_ad (m_dot_br, H_u, h_b, m_dot_l, cp_l, t_l, m_dot_rg, cp_rg)
```

```
      {return (m_dot_br*(H_u+h_b)+m_dot_l*cp_l*(t_l-273.15))/(m_dot_rg*cp_rg)+273.15;}

   @eq cont t_g (a, t_FA)
      {return a*t_FA;}

   @eq cont a (z, epsilon)
      {return z*pow(epsilon, -0.25);}

   @eq cont Ko (m_dot_rg, cp_rg, z, C_S, A, f, ntubes, t_ad)
      {return (m_dot_rg*cp_rg)/(pow (z, 4.0)*C_S*A*f*ntubes*pow (t_ad, 3.0));}

   @eq cont t_FA (t_FA, t_ad, Ko)
      {return t_ad*(1-1/Ko*pow ((t_FA/t_ad), 4.0));}

@end
```

B.3.3 „Waermedurchgang.smi“

Die mathematische Modellierung der Wärmedurchgangsklassen steht in Kapitel 2.4.4 ab Seite 17.

```
#include "Wand.smd"
#include "Wue.smd"

#include "consts.h"
#include "arraydefs.h"

@interface Waermedurchgang_cuts : SmileObject{        // dQ=f(T)

   @export
      double T1[ND]   [doc: "temperature 1", unit: "K", min: 0.0, max: 5000.0];
      double T2[ND]   [doc: "temperature 2", unit: "K", min: 0.0, max: 5000.0];
      double dQ1[ND]  [eq, doc: "heat flux 1", unit: "W"];  //eq -> verhindert falsche connects
      double dQ2[ND]  [eq, doc: "heat flux 2", unit: "W"];
      double A        [doc: "area", unit: "m^2"];
   }

@end

@implementation Waermedurchgang_cuts

   @eq cont dQ1[i] (time)
      {return 0.0;}

   @eq cont dQ2[i] (time)
      {return 0.0;}

@end


@interface WaermedurchgangConstK : Waermedurchgang_cuts{

   @export
      double dQ1[ND] [eq, doc: "heat flux 1", unit: "W"];
      double dQ2[ND] [eq, doc: "heat flux 2", unit: "W"];

   @protected
```

Klassenname	Elternklasse	@export-Variable	Gleichung
Waermedurchgang_cuts	SmileObject	T1[ND] T2[ND] dQ1[ND] dQ2[ND] A	
WaermedurchgangConstK	Waermedurchgang_cuts	dQ1[ND] dQ2[ND]	× ×
WaermedurchgangRohr-WueKonvHeatStoreWueKonv	Waermedurchgang_cuts		
WaermedurchgangRohr-WueKonvWaermeleitungWueKonv	Waermedurchgang_cuts		

Tabelle 12: @export-Variablen der Wärmedurchgangsklassen

```
      double k        [doc: "Waermedurchgangskoeffizient", unit: "W/m^2/K"];
      double dQ[ND]   [eq, doc: "heat flux", unit: "W"];
   }

   @init{
      k = 100.0;
   }

@end

@implementation WaermedurchgangConstK

   @eq cont dQ[i] (T1[i], T2[i], k, A)
      {return k*A*(T1[i]-T2[i]);}

   @eq cont dQ1[i] (dQ[i])
      {return -dQ[i];}

   @eq cont dQ2[i] (dQ[i])
      {return dQ[i];}

@end



@interface WaermedurchgangRohrWueKonvHeatStoreWueKonv : Waermedurchgang_cuts{

   @protected
      WueKonvektion*          wuekonv1;    // dQ=f(T)
      ThinWallTubes*          wand;        // T=f(dQ)
      WueKonvektion*          wuekonv2;    // dQ=f(T)
   }

@end

@implementation WaermedurchgangRohrWueKonvHeatStoreWueKonv

   @component wuekonv1{}
```

```
   @component wand{}

   @component wuekonv2{}

   @connect{
            wuekonv1.dQ[i]     = wand.dQ1[i];
            wuekonv1.t_wall[i] = wand.T1[i];
            wand.dQ2[i]        = wuekonv2.dQ[i];       // Gleichstrom
            wand.T2[i]         = wuekonv2.t_wall[i];   // Gleichstrom

            T1[i]              = wuekonv1.t_fluid[i];
            T2[i]              = wuekonv2.t_fluid[i];
            dQ1[i]             = wuekonv1.dQ[i];
            dQ2[i]             = wuekonv2.dQ[i];

            A                  = wuekonv1.A     =   wand.A   = wuekonv2.A;
           }

@end



@interface WaermedurchgangRohrWueKonvWaermeleitungWueKonv : Waermedurchgang_cuts{

   @protected
      WueKonvektion*          wuekonv1;   // dQ=f(T)
      WaermeleitungRohr*      wand;       // T=f(dQ)
      WueKonvektion*          wuekonv2;   // dQ=f(T)
   }

@end

@implementation WaermedurchgangRohrWueKonvWaermeleitungWueKonv

   @component wuekonv1{}

   @component wand{}

   @component wuekonv2{}

   @connect{
            wuekonv1.dQ[i]     = wand.dQ1[i];
            wuekonv1.t_wall[i] = wand.T1[i];
            wand.dQ2[i]        = wuekonv2.dQ[i];       // Gleichstrom
            wand.T2[i]         = wuekonv2.t_wall[i];   // Gleichstrom

            T1[i]              = wuekonv1.t_fluid[i];
            T2[i]              = wuekonv2.t_fluid[i];
            dQ1[i]             = wuekonv1.dQ[i];
            dQ2[i]             = wuekonv2.dQ[i];

            A                  = wuekonv1.A     =   wand.A   = wuekonv2.A;
           }

@end
```

B.4 Wärmeleitung und -speicherung

Klassenname	Elternklasse	`@export`-Variable	Gleichung

Wand_cuts	SmileObject	`T1[ND]`
		`T2[ND]`
		`dQ1[ND]`
		`dQ2[ND]`
		`A`	

DiscreteHeatStore	Wand_cuts	`T1[ND]` `T2[ND]`	× ×
ThinWall	DiscreteHeatStore		
ThinWallTubes	ThinWall	`A`	×
Waermeleitung	Wand_cuts	`T1[ND]` `T2[ND]`	× ×
WaermeleitungRohr	Waermeleitung	`A`	×

Tabelle 13: `@export`-Variablen der Wärmeleitungs- und -speicherungsklassen

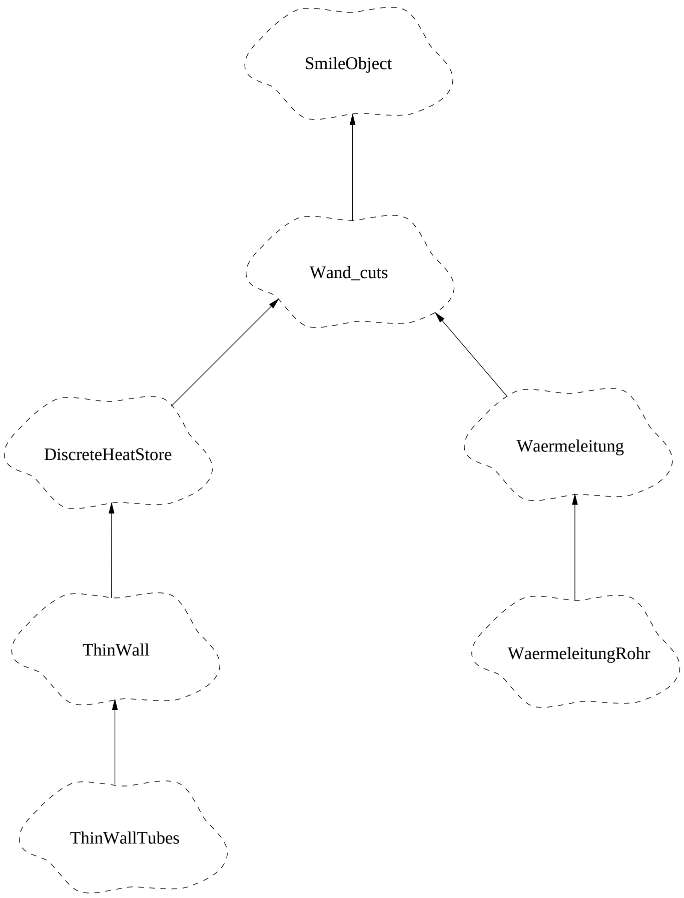

Abbildung 22: Klassenstruktur Wärmeleitung und -speicherung

B.4.1 „Wand.smi“

Die mathematische Modellierung der Wärmeleitungsklassen ist in Kapitel 2.4.1 ab Seite 10 zu finden, die der Wärmespeicherklassen in Kapitel 2.5 ab Seite 17.

```
#include "consts.h"
#include "arraydefs.h"
#include "zeitkonstanten.h"

@interface Wand_cuts : SmileObject{

    @export
        double T1[ND]   [eq, doc: "temperature 1", unit: "K", min: 0.0, max: 5000.0];
                     //eq -> verhindert falsche connects
        double T2[ND]   [eq, doc: "temperature 2", unit: "K", min: 0.0, max: 5000.0];
        double dQ1[ND]  [doc: "heat flux 1", unit: "W"];
        double dQ2[ND]  [doc: "heat flux 2", unit: "W"];
        double A        [doc: "area", unit: "m^2", min: 0.0];
    }

@end

@implementation Wand_cuts

   @eq cont T1[i] (time)
      {return 0.0;}

   @eq cont T2[i] (time)
      {return 0.0;}

@end



@interface DiscreteHeatStore : Wand_cuts
{
    @export
       double T1[ND] [eq, doc: "temperature", unit: "K", min: 0.0, max: 5000.0];
       double T2[ND] [eq, doc: "temperature", unit: "K", min: 0.0, max: 5000.0];

    @protected
        double n      [doc: "no of discrete elements"];
        double T[ND]  [eq, doc: "temperature", unit: "K", min: 0.0, max: 5000.0];
        double m      [doc: "mass per element", unit: "kg", min: 0.0];
        double cp     [doc: "heat capacity", unit: "J/kg/K"];
        double dQ[ND] [doc: "sum of heat fluxes", unit: "W"];
}
  @init{
      int i;
      n = ND;
      for(i=0;i<ND;i++){
          T[i] = 350.0;
          dQ[i] = 0.0;
      }
      m = 1.0;
      cp = 0.5;
}
@end

@implementation DiscreteHeatStore
```

```
  @eq diff T[i] (dQ[i], m, cp){
     return dQ[i] / (m * cp);}

  @eq cont T1[i] (T[i])
     {return T[i];}

  @eq cont T2[i] (T[i])
     {return T[i];}

@end




@interface ThinWall : DiscreteHeatStore
{
    @protected
        double rho    [doc: "density", unit: "kg/m^3"];
        double b      [doc: "thickness", unit: "m"];
        double m      [eq, doc: "mass per element", unit: "kg", min: 0.0];
        double dQ[ND] [eq, doc: "sum of heat fluxes", unit: "W"];
}
    @init {
        A   = 1.0;
        b   = 0.008;
        rho = 7900.0;
    }
@end

@implementation ThinWall

  @eq cont m (rho, A, b, n){
      return rho * A * b / n;
  }
  @eq cont dQ[i] (dQ1[i], dQ2[i]){
      return -dQ1[i]-dQ2[i];
  }
@end




@interface ThinWallTubes : ThinWall
{
    @export
        double A      [eq, doc: "area", unit: "m^2", min: 0.0];

    @protected
        double n_tubes [doc: "no of tubes"];
        double dm       [doc: "diameter", unit: "m"];
        double l        [doc: "length", unit: "m"];
        double m        [eq, doc: "mass per element", unit: "kg", min: 0.0];
}
    @init {
        n_tubes = 1.0;
        dm = 0.05;
        l = 1.0;
    }
@end

@implementation ThinWallTubes

  @eq cont A (dm, l, n_tubes) {
      return PI * dm * l * n_tubes;
```

```
  }
  @eq cont m (dm, b, l, rho, n_tubes, n) {
      return n_tubes * l * PI * dm * b * rho / n;
  }
@end



@interface Waermeleitung : Wand_cuts{

      /*smile-doc
        - description   : instationaere Waermeleitung (impliziert Waermespeicherung!)
        - type          :
        - assumptions   : ebene Wand (duenne Rohrwand), kein Quellterm
        - limitations   :
        - reference     : {PatankarS80}, S. 41ff
        - maturity      :
        - numerical tips:
        - author        : Oliver Czygan <Czygan@tu-harburg.d400.de>
        - maintainer    :
        - creation date : Jan 10 1997
        - last modified : $__mtime$
        - version       : $__self$
        smile-doc*/

   @export
      double T1[ND]       [eq, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
      double T2[ND]       [eq, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];

   @protected
      double T[ND][ND_WL] [eq, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];

/* T:      erster Index: Diskretisierung in Rohrachsrichtung         (i) */
/*        zweiter Index: Diskretisierung in wandsenkrechter Richtung (j) */

      double q_dot_1[ND]  [eq, doc: "Waermeflaechenleistung", unit: "W/m^2"];
      double q_dot_2[ND]  [eq, doc: "Waermeflaechenleistung", unit: "W/m^2"];
      double lambda       [doc: "Waermeleitkoeffizient", unit: "W/m/K", min: 0.0];
      double rho          [doc: "Dichte", unit: "kg/m^3"];
      double c_p          [doc: "spezif. Waermekapazitaet", unit: "J/kg/K"];
      double a            [eq, doc: "Temperaturleitfaehigkeit", unit: "m^2/s",
                           min: 0.0];
      double s            [doc: "Wandstaerke", unit: "m"];
      double delta_r      [eq, doc: "Wandstaerke eines diskreten Elements",
                           unit: "m", min: 0.0];
      }

   @init
      {
      int i;
      int j;
      for (j=0; j<ND_WL; j++)
         {for (i=0; i<ND; i++)
           {T[i][j] = 300.0;}}

      for (i=0; i<ND; i++)
         {
          dQ1[i] = 0.0;
          dQ2[i] = 0.0;
         }

      lambda = 15.0;
      rho    = 8000.0;
```

```
      c_p    = 477.0;
      s       = 0.01;
      }

@end

@implementation Waermeleitung

   @eq cont delta_r (s)
      {return s/(ND_WL-1);}

   @eq cont a (lambda, c_p, rho)
      {return lambda/c_p/rho;}

   @eq cont q_dot_1[i] (dQ1[i], A)
      {return dQ1[i]/A*ND);}

   @eq cont q_dot_2[i] (dQ2[i], A)
      {return dQ2[i]/A*ND);}

   @eq diff T[i][j:1::ND_WL-2] (a, delta_r, T[i][j-1], T[i][j], T[i][j+1])
      {return a/delta_r/delta_r*(T[i][j+1]+T[i][j-1]-2*T[i][j]);}

/* Die beiden folgenden Gleichungen (= Randbedingungen) muessen
   fuer Smile auch als Dgls formuliert werden (ueber kuenst-
   lichen Integrator).                                            */

   @eq diff T[i][0] (T[i][1], T[i][0], delta_r, lambda, q_dot_1[i])
      {return 1/TAU_MITTEL*(T[i][1]-T[i][0]-delta_r/lambda*q_dot_1[i]);}

   @eq diff T[i][ND_WL-1] (T[i][ND_WL-1], T[i][ND_WL-2], delta_r, lambda,
                           q_dot_2[i])
      {return 1/TAU_MITTEL*(T[i][ND_WL-2]-T[i][ND_WL-1]-delta_r/lambda*
              q_dot_2[i]);}

   @eq cont T1[i] (T[i][0])
      {return T[i][0];}

   @eq cont T2[i] (T[i][ND_WL-1])
      {return T[i][ND_WL-1];}

@end




@interface WaermeleitungRohr : Waermeleitung{

        @export
           double A  [eq, doc: "Flaeche", unit: "m^2", min: 0.0];

        @protected
           double di [doc: "Innendurchmesser", unit: "m"];
           double da [doc: "Aussendurchmesser", unit: "m"];
           double s  [eq, doc: "Wandstaerke", unit: "m", min: 0.0];
           double l  [doc: "Rohrlaenge", unit: "1"];
           double f  [doc: "Wandflaechenfaktor", unit: "1"];
           /*  beruecksichtigt z.B. Verschmutzung, einseitige
               Waermebeaufschlagung oder
               Flaechenvergroesserung durch Flossen  */
           double n_tubes [doc: "Anzahl der Rohre", unit: "1"];
        }

        @init{
```

```
          di = 20.0e-03;
          da = 24.0e-03;
          s  = 2.0e-03;
          l  = 1.0;
          A  = 1.0;
          f  = 1.0;
          n_tubes = 1.0;
          }

@end

@implementation WaermeleitungRohr

        @eq cont s (di, da)
           {return (da-di)/2;}

        @eq cont A (n_tubes, di, da, f, l)
           {return f*n_tubes*l*(da+di)/2*PI;}

@end
```

B.5 Massespeicherung

B.5.1 „Druckspeicher.smi“

Die mathematische Modellierung des Druckspeichers ist in Kapitel 4.1.3 ab Seite 27 zu finden.

```
#include "arraydefs.h"

@interface FlowCut : SmileObject
{
    @export //In
        double m_dot_1 [doc: "massflow", unit: "kg/s"];
        double p_1 [doc: "pressure", unit: "Pa"];
        double rho_1 [doc: "density", min: "0.0", max: "25.0e3", unit: "kg/m^3"];

    @export //Out
        double m_dot_2 [doc: "massflow", unit: "kg/s"];\
        double p_2 [doc: "pressure", unit: "Pa"];
        double rho_2 [doc: "density", min: "0.0", max: "25.0e3", unit: "kg/m^3"];

}
@end

@implementation FlowCut

@end




@interface PressureStore : FlowCut
{
   @export
        double p [eq, doc: "pressure", unit: "Pa"];
        double m [eq, doc: "mass of liquid", unit: "kg"];
        double rho [eq, doc: "density", min: "0.0", max: "25.0e3", unit: "kg/m^3"];
        double elastic [doc: "constant for numerical reason", unit: "J"];
```

Klassenname	Elternklasse	@export-Variable	Gleichung
FlowCut	SmileObject	m_dot_1 p_1 rho_1 m_dot_2 p_2 rho_2	
PressureStore	FlowCut	p m rho elastic V_0	× × ×

Tabelle 14: @export-Variablen der Massespeicherungsklassen

```
        double V_0 [doc: "elacstic volume", unit: "m^3"];
}
    @init {
        m = 100.0;
        p = 1.0e5;
        V_0 = 1.0;
        elastic = 90000.0; //increase for higher pressures
        }
@end

@implementation PressureStore

    @eq diff m (m_dot_1, m_dot_2)
    { return m_dot_1 - m_dot_2; }

    @eq cont rho (rho_1, rho_2)
    { return (rho_1 + rho_2) * 0.5; }

    @eq cont p (V_0, elastic, m, rho)
    { return elastic / (V_0 - m/rho); }

@end
```

B.6 Stoffwertefunktionen

B.6.1 „SWFDampf.smi“

```
#include <steam.h>
#include "arraydefs.h"

@interface SWFPUndTAusHUndRho_cuts : SmileObject{

   @export
      double p[ND]         [eq, doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
                                  // eq -> verhindert falsche connects
      double T[ND]         [eq, hist, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];
```

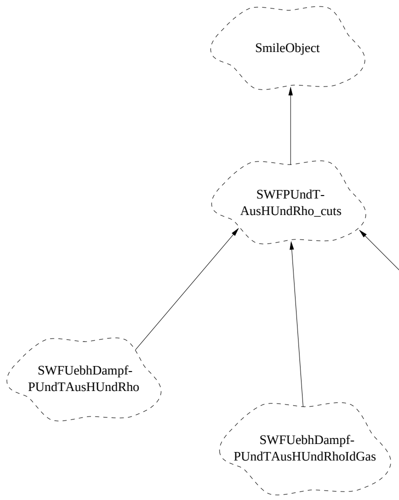

Abbildung 23: Klassenstruktur Stoffwertefunktionen

Klassenname	Elternklasse	`@export`-Variable	Gleichung
SWFPUndTAusHUndRho_cuts	SmileObject	`p[ND]`	
		`T[ND]`	
		`h[ND]`	
		`rho[ND]`	
SWFNassdampfPUndTAusHUndRho	SWFPUndTAusHUndRho_cuts	`p[ND]`	×
		`T[ND]`	×
SWFUebhDampfPUndTAusHUndRho	SWFPUndTAusHUndRho_cuts	`p[ND]`	×
		`T[ND]`	×
SWFUebhDampfPUndTAusHUndRhoIdGas	SWFPUndTAusHUndRho_cuts	`p[ND]`	×
		`T[ND]`	×

Tabelle 15: `@export`-Variablen der Stoffwertefunktionsklassen

```
      double h[ND]         [doc: "Enthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double rho[ND]       [doc: "Dichte", unit: "kg/m^3", min: 0.0, max: 25.0e+03];

   }

@end

@implementation SWFPUndTAusHUndRho_cuts

   @eq cont p[i] (time)
      {return 0.0;}

   @eq cont T[i] (time)
      {return 0.0;}

@end



@interface SWFNassdampfPUndTAusHUndRho : SWFPUndTAusHUndRho_cuts{

   @export
      double p[ND]     [eq, doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      double T[ND]     [eq, hist, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];

   @protected
      double x[ND]     [eq, doc: "Dampfgehalt", unit: "1", min: 0.0, max: 1.0];
      double r[ND]     [eq, doc: "Verdampfungswaerme", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double h_1[ND]   [eq, doc: "Enthalpie fuer x=0", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double h_2[ND]   [eq, doc: "Enthalpie fuer x=1", unit: "J/kg", min: 0.0, max: 1.0e+07];
      double rho_1[ND] [eq, doc: "Dichte fuer x=0", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double rho_2[ND] [eq, doc: "Dichte fuer x=1", unit: "kg/m^3", min: 0.0, max: 25.0e+03];
      double nassdampfcheck[ND]   [eq, doc: "Schalter fuer Fluessigkeit <-> Nassdampf",
                                   unit: "1"];
   }

   @init{
         int i;
         for (i=0; i<ND; i++)
           {
            p[i]   = 12.0e+05;
             // Hiermit wird auch T[i][|hist|] gesetzt.
            T[i]   = 600.0;
            h[i]   = 2782.7e+03;
            rho[i] = 6.127;
           }
        }

@end

@implementation SWFNassdampfPUndTAusHUndRho

   @eq cont T[j] (rho[j], h[j])
      {
     double prop[14];
     double T_guess;

       // Der Schätzwert für T sollte nicht zu klein oder zu groß sein.
      T_guess = T[j][|last|];

       if      (T_guess < 300.0)  {T_guess = 300.0;}
       else if (T_guess > 1000.0) {T_guess = 1000.0;}
```

```
        whd (h[j], rho[j], T_guess, 1.0e-12, prop);

        if (prop[IDX_ERROR])
           {
            printf ("%f: error %d bei h=%f \t rho=%f\n", time, (int) j, h[j], rho[j]);
            return 1.0 / 0.0;
           }
        return prop[IDX_T];
     }

@eq discrete nassdampfcheck[j] (h[j], h_1[j])
   {
      if (h[j] < h_1[j]) {return 0.0;}             /* Fluessigkeitsgebiet */
      else               {return 1.0;}             /* Nassdampfgebiet     */
   }

@eq cont p[j] (h[j], rho[j], T[j])
   {
  double prop[14];
  double T_guess;

    // Der Schätzwert für T sollte nicht zu klein oder zu groß sein.
   T_guess = T[j][|last|];

    if      (T_guess < 300.0)  {T_guess = 300.0;}
    else if (T_guess > 1000.0) {T_guess = 1000.0;}

       whd (h[j], rho[j], T_guess, 1.0e-12, prop);

       if (prop[IDX_ERROR])
          {
           printf ("%f: error %d bei h=%f \t rho=%f\n", time, (int) j, h[j], rho[j]);
           return 1.0 / 0.0;
          }
       return prop[IDX_P];
   }

@eq cont h_1[j] (p[j]){
   double liq[14];
   double vap[14];

   sat_p (p[j], liq, vap);
   return liq[IDX_H];
   }

@eq cont h_2[j] (p[j]){
   double liq[14];
   double vap[14];

   sat_p (p[j], liq, vap);
   return vap[IDX_H];
   }

@eq cont rho_1[j] (p[j]){
   double liq[14];
   double vap[14];

   sat_p (p[j], liq, vap);
   return liq[IDX_D];
   }

@eq cont rho_2[j] (p[j]){
   double liq[14];
   double vap[14];
```

```
      sat_p (p[j], liq, vap);
      return vap[IDX_D];
      }

   @eq cont r[j] (h_1[j], h_2[j])
      {return h_2[j]-h_1[j];}

   @eq cont x[j] (h[j], h_1[j], r[j])
      {
       return (h[j]-h_1[j])/r[j];
      }

@end




@interface SWFUebhDampfPUndTAusHUndRho : SWFPUndTAusHUndRho_cuts{

   @export
      double p[ND]         [eq, doc: "Druck", unit: "Pa"];
      double T[ND]         [eq, hist, doc: "Temperatur", unit: "K"];
   }

   @init{
         int i;
         for (i=0; i<ND; i++)
           {
            p[i]   = 12.0e+05;
             // Hiermit wird auch T[i][|hist|] gesetzt.
            T[i]   = 600.0;
            h[i]   = 2782.7e+03;
            rho[i] = 6.127;
           }
        }

@end

@implementation SWFUebhDampfPUndTAusHUndRho

   @eq cont T[j] (h[j], rho[j]){
      double vek[14];
      double T_guess;

       // Der Schätzwert für T sollte nicht zu klein oder zu groß sein.
      T_guess = T[j][|last|];
      if (T_guess < 300.0) {
                             T_guess = 300.0;
                            }
      else if (T_guess > 1000.0) {
                                   T_guess = 1000.0;
                                  }

      whd (h[j], rho[j], T_guess, 1.0e-12, vek);
      if (vek[IDX_ERROR]) {
       printf ("%f: error %d bei h=%f \t rho=%f\n", time, (int) j, h[j], rho[j]);
       return 1.0 / 0.0;
                          }
      return vek[IDX_T];}

   @eq cont p[j] (h[j], rho[j], T[j]){
      double vek[14];
      double T_guess;
```

```
     // Der Schätzwert für T sollte nicht zu klein oder zu groß sein.
    T_guess = T[j][|last|];
    if (T_guess < 300.0) {
                         T_guess = 300.0;
                        }
    else if (T_guess > 1000.0) {
                               T_guess = 1000.0;
                              }

    whd (h[j], rho[j], T_guess, 1.0e-12, vek);
    if (vek[IDX_ERROR]) {
     printf ("%f: error %d bei h=%f \t rho=%f\n", time, (int) j, h[j], rho[j]);
     return 1.0/0.0;
                       }
    return vek[IDX_P];}

@end




@interface SWFUebhDampfPUndTAusHUndRhoIdGas : SWFPUndTAusHUndRho_cuts{

  @export
    double p[ND]          [eq, doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
    double T[ND]          [eq, hist, doc: "Temperatur", unit: "K", min: 0.0, max: 5000.0];

  @protected
    double cp       [doc: "spezif. Waermekapazitaet bei konst. Druck", unit: "J/kg/K"];
    double R        [doc: "Gaskonstante", unit: "J/kg/K"];
    double T_S[ND] [eq, doc: "Saettigungstemperatur", unit: "K", min: 0.0, max: 5000.0];
    double h_2[ND] [eq, doc: "Sattdampfenthalpie", unit: "J/kg", min: 0.0, max: 1.0e+07];
  }

  @init{
      int i;
      for (i=0; i<ND; i++)
        {
         p[i]   = 12.0e+05;
          // Hiermit wird auch T[i][|hist|] gesetzt.
         T[i]   = 600.0;
         h[i]   = 2782.7e+03;
         rho[i] = 6.127;
         T_S[i] = 186.05+273.15;
         h_2[i] = 2781.3e+03;
        }
      cp  = 2.2e+03;
      R   = 462.0;
     }

@end

@implementation SWFUebhDampfPUndTAusHUndRhoIdGas

  @eq cont p[j] (R, T[j], rho[j])
    {return R*T[j]*rho[j];}

  @eq cont T[j] (h[j], h_2[j], cp, T_S[j])
    {return (h[j]-h_2[j])/cp+T_S[j];}

  @eq cont T_S[i] (p[i])
    {double liq[14];
     double vap[14];
```

```
      sat_p (p[i], liq, vap);
      return vap[IDX_T];}

   @eq cont h_2[i] (p[i])
     {double liq[14];
      double vap[14];

      sat_p (p[i], liq, vap);
      return vap[IDX_H];}

@end
```

B.7 Dampferzeugerrohre

Klassenname	Elternklasse	@export-Variable	Gleichung
RgWasserWAT	SmileObject	m_dot_rg m_dot_wasser_in m_dot_wasser_out t_rg_in p_rg_in rho_rg_in t_wasser_in h_wasser_in p_wasser_in rho_wasser_in t_rg_out p_rg_out rho_rg_out t_wasser_out h_wasser_out p_wasser_out rho_wasser_out	
Ecorohr	RgWasserWAT		
Verdampferrohr	RgWasserWAT		
Ueberhitzerrohr	RgWasserWAT		
Einspritzkuehler_cuts	SmileObject	m_dot_in m_dot_out m_dot_w h_in h_out h_w p	
EinspritzkuehlerOhneSpeicher	Einspritzkuehler_cuts	m_dot_in h_out t_in t_out t_w	× × × × ×

Klassenname	Elternklasse	@export-Variable	Gleichung
Drossel	SmileObject	m_dot_1	×
		m_dot_2	×
		p_1	
		p_2	
		h_1	
		h_2	×

Tabelle 16: @export-Variablen der Klassen der Dampferzeugereinzelkomponenten

B.7.1 „RgWasserWAT.smi“

```
#include "arraydefs.h"

@interface RgWasserWAT : SmileObject{

  @export
    double m_dot_rg          [doc: "Rauchgasmassenstrom", unit: "kg/s", min: 0.0,
                               max: 1.0e+06];
    double m_dot_wasser_in   [doc: "Eintrittswassermassenstrom", unit: "kg/s",
                               min: 0.0, max: 1.0e+06];
    double m_dot_wasser_out  [doc: "Austrittswassermassenstrom", unit: "kg/s",
                               min: 0.0, max: 1.0e+06];

    double t_rg_in       [doc: "Rauchgaseintrittstemperatur", unit: "K", min: 0.0,
                          max: 5000.0];
    double p_rg_in       [doc: "Rauchgaseintrittsdruck", unit: "Pa", min: 0.0,
                          max: 1000.0e+05];
    double rho_rg_in     [doc: "Rauchgaseintrittsdichte", unit: "kg/m^3", min: 0.0,
                          max: 25.0e+03];
    double t_wasser_in   [doc: "Wassereintrittstemperatur", unit: "K", min: 0.0,
                          max: 5000.0];
    double h_wasser_in   [doc: "Wassereintrittsenthalpie", unit: "J/kg", min: 0.0,
                           max: 1.0e+07];
    double p_wasser_in   [doc: "Wassereintrittsdruck", unit: "Pa", min: 0.0,
                           max: 1000.0e+05];
    double rho_wasser_in [doc: "Wassereintrittsdichte", unit: "kg/m^3", min: 0.0,
                           max: 25.0e+03];

    double t_rg_out       [doc: "Rauchgasaustrittstemperatur", unit: "K", min: 0.0,
                           max: 5000.0];
    double p_rg_out       [doc: "Rauchgasaustrittsdruck", unit: "Pa", min: 0.0,
                           max: 1000.0e+05];
    double rho_rg_out     [doc: "Rauchgasaustrittsdichte", unit: "kg/m^3", min: 0.0,
                           max: 25.0e+03];
    double t_wasser_out   [doc: "Wasseraustrittstemperatur", unit: "K", min: 0.0,
                           max: 5000.0];
    double h_wasser_out   [doc: "Wasseraustrittsenthalpie", unit: "J/kg", min: 0.0,
                           max: 1.0e+07];
    double p_wasser_out   [doc: "Wasseraustrittsdruck", unit: "Pa", min: 0.0,
                           max: 1000.0e+05];
    double rho_wasser_out [doc: "Wasseraustrittsdichte", unit: "kg/m^3", min: 0.0,
                           max: 25.0e+03];
    }

  @init{
```

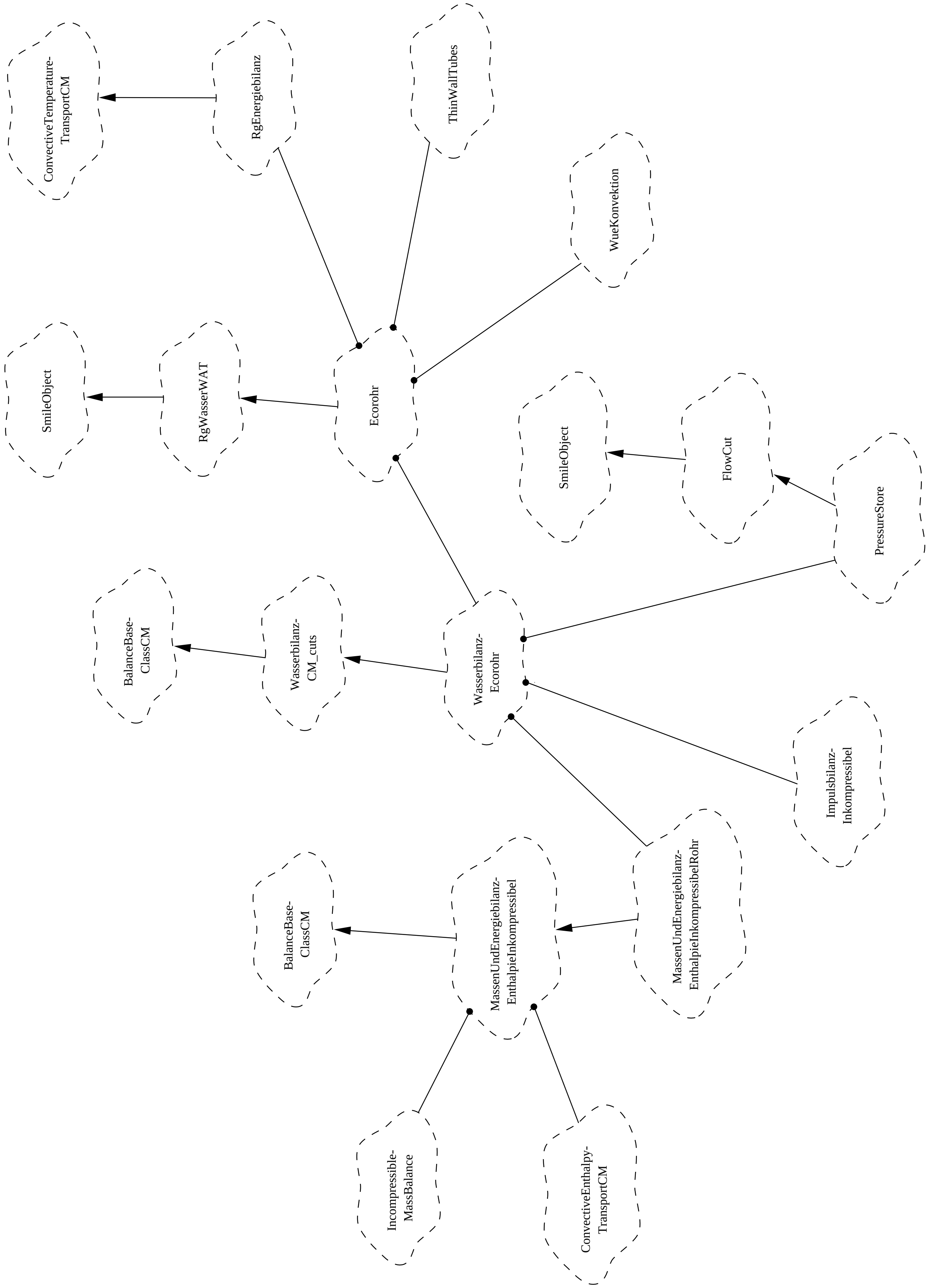

Abbildung 24: Klassenstruktur Ecorohr

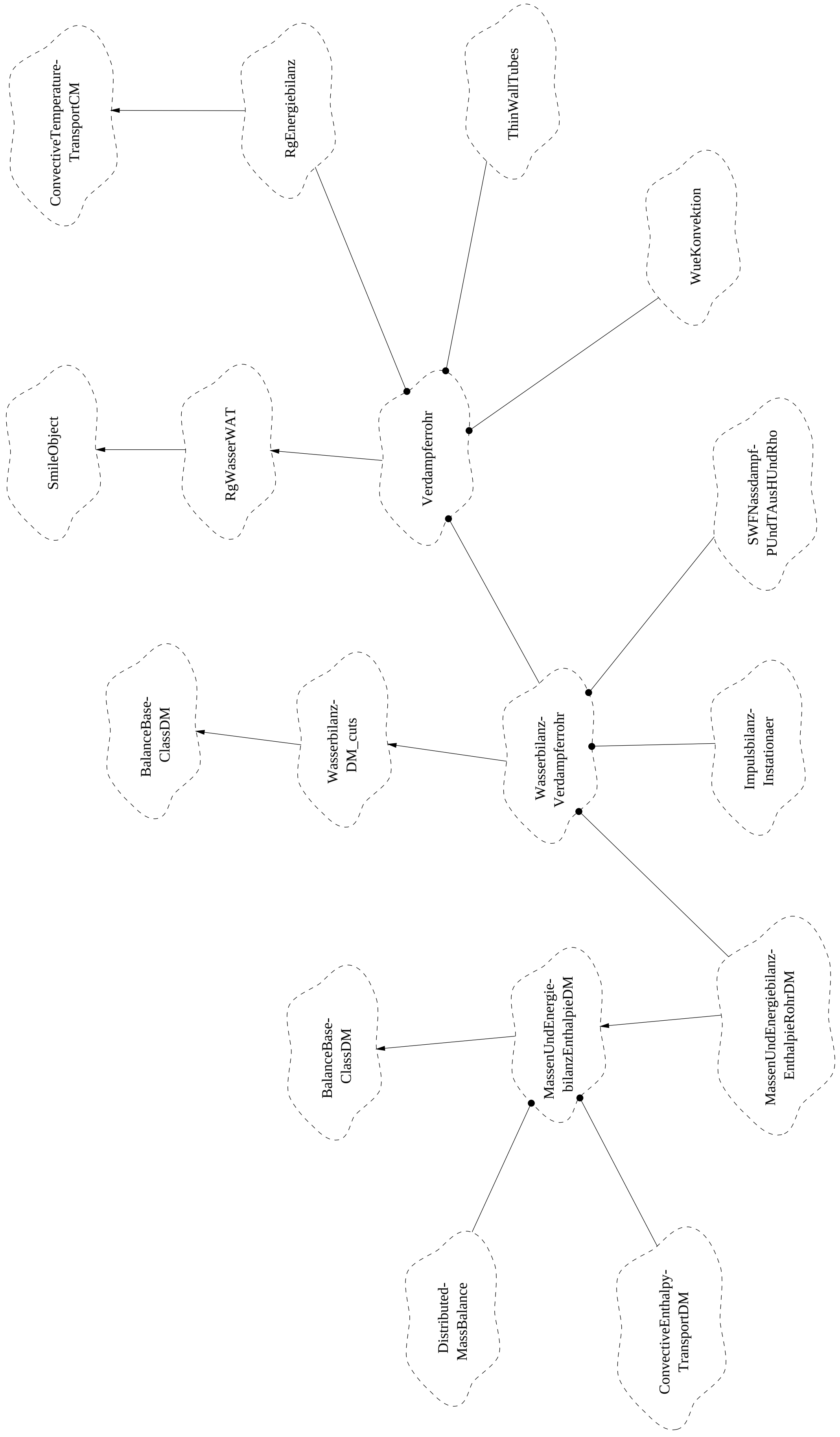

Abbildung 25: Klassenstruktur Verdampferrohr

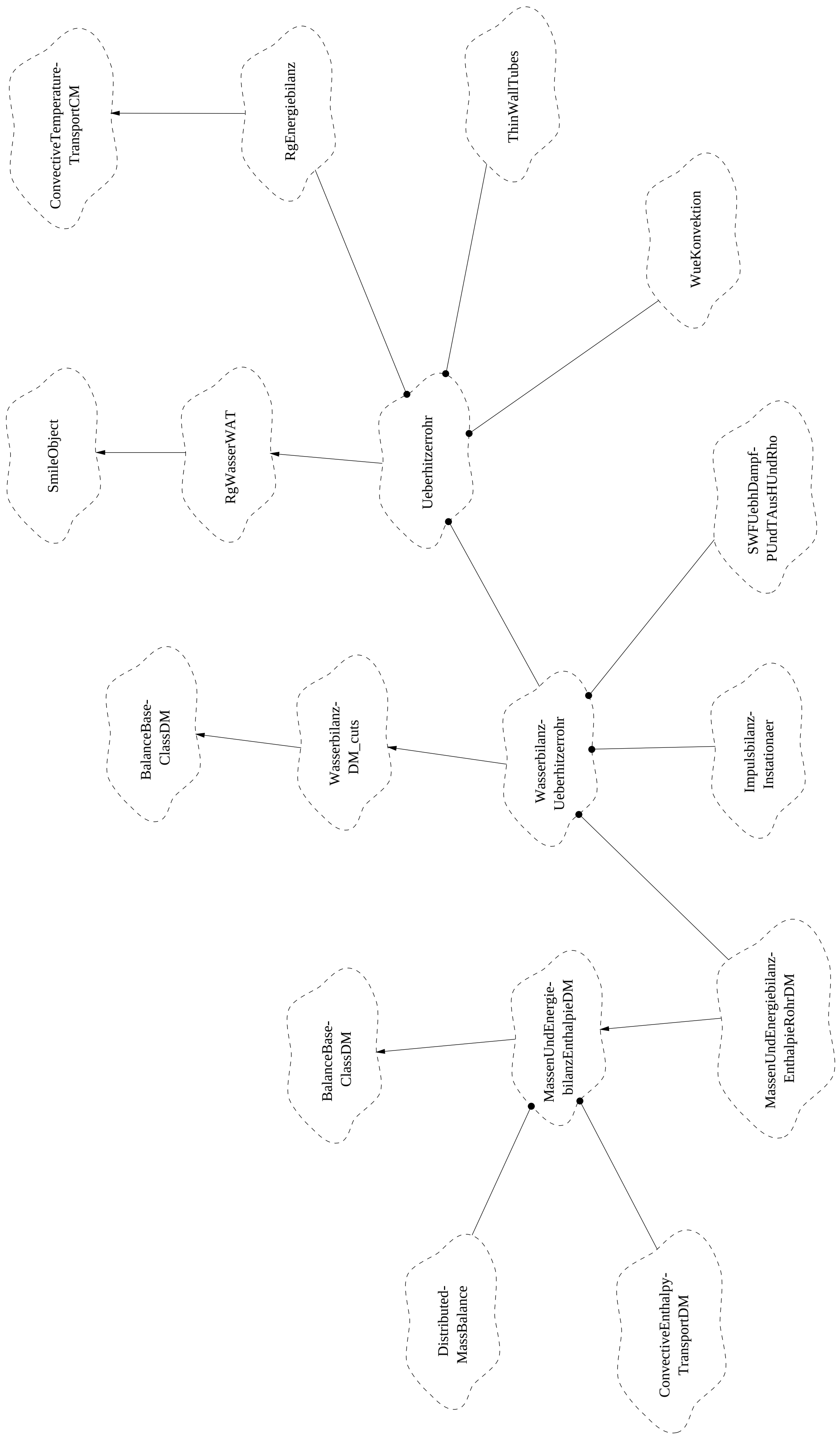

Abbildung 26: Klassenstruktur Überhitzerrohr

```
        m_dot_rg = m_dot_wasser_in = m_dot_wasser_out = t_rg_in = p_rg_in = rho_rg_in =
        t_wasser_in = h_wasser_in = p_wasser_in = rho_wasser_in = t_rg_out = p_rg_out =
        rho_rg_out = t_wasser_out = h_wasser_out = p_wasser_out = rho_wasser_out = 0.0;
       }

@end

@implementation RgWasserWAT

@end
```

B.7.2 „Ecorohr.smi“

```
#include "RgWasserWAT.smd"
#include "Wasserbilanz.smd"
#include "RgEnergiebilanz.smd"
#include "Wue.smd"
#include "Wand.smd"

#include "arraydefs.h"
#include "zeitkonstanten.h"

@interface Ecorohr : RgWasserWAT{

   @protected
      WasserbilanzEcorohr*                        wasser;
//      WueKonvRohrInnen*                          wue_innen;
      WueKonvektion*                              wue_innen;
//       WaermeleitungRohr*                          wand;
      ThinWallTubes*                              wand;
//       WueStrahlungUndKonvektionRohr*              wue_aussen;
      WueKonvektion*                              wue_aussen;
      RgEnergiebilanz*                            rg;

   @export
      double delta_t      [eq, doc: "Zeitschrittweite", unit: "s", min: 0.0];
      double zeit         [eq, hist, doc: "Zeit", unit: "s", min: 0.0];
      double Q_dot_wasser [eq, doc: "Wasserwaermeleistung", unit: "W"];
      double Q_dot_rg     [eq, doc: "Rauchgaswaermeleistung", unit: "W"];
      double cp           [doc: "spezif. Waermekapazitaet Rg", unit: "J/kg/K"];
   }

   @init{
      m_dot_wasser_in  = 2.5116;
      m_dot_wasser_out = 2.5116;
      cp               = 1131.1;
      zeit             = 0.0;
     }

@end

@implementation Ecorohr

   @eq cont zeit (time)
      {return time;}

   @eq cont delta_t (zeit)
      {return zeit-zeit[|last|];}

   @eq cont Q_dot_wasser (m_dot_wasser_in, m_dot_wasser_out, h_wasser_in, h_wasser_out)
      {return m_dot_wasser_out*h_wasser_out-m_dot_wasser_in*h_wasser_in;}
```

```
   @eq cont Q_dot_rg (m_dot_rg, cp, t_rg_in, t_rg_out)
      {return m_dot_rg*cp*(t_rg_in-t_rg_out);}

   @component wasser{ /* Initialisierungen in 'Wasserbilanz.smi' */ }

//    @component wue_innen{                   /* WueKonvRohrInnen */
//          int i;
//          for (i=0; i<ND; i++)
//             {
//           t_fluid[i] = 251.48 + 273.15;
//              t_wall[i]  = 700.0;
//             }
//          di      = 19.8e-03;
//          da      = 25.0e-03;
//          l       = 27.180;
//          rho     = 20.50;
//          Pr      = 1.2;
//          eta     = 10.0e-06;
//          lambda  = 20.0;
//          c       = 12.44;
//          n_tubes = 32;
//          xi      = 0.02;
//         }

   @component wue_innen{                      /* WueKonvektion */
         int i;
         for (i=0; i<ND; i++)
            {
             t_fluid[i] =  105.0 + 273.15;
             t_wall[i]  = 600.0;
            }
         alpha   = 2971.44;
         A       = 1215.38;
         f       = 1.0;
        }

//    @component wand{                  /* Waermeleitung */
//          int i;
//          int j;
//          for (i=0; i<ND_WL; i++)
//             {
//           for (j=0; j<ND; j++)
//                 {T[j][i] = 700.0;
//                  T1[j]   = 700.0;
//                  T2[j]   = 700.0;}
//             }
//          lambda  = 15.0;
//          rho     = 8000.0;
//          c_p     = 477.0;
//          di      = 19.8e-03;
//          da      = 25.0e-03;
//          l       = 27.180;
//          f       = 1.0;
//          n_tubes = 32;
//         }

   @component wand{                  /* ThinWallTubes */
         int j;
         for (j=0; j<ND; j++)
            {T[j]    = 600.0;
             T1[j]   = 600.0;
             T2[j]   = 600.0;}
         rho     = 8000.0;
```

```
         cp      = 477.0;
         dm      = 22.4e-03;
         b       = 2.6e-03;
         l       = 72.8;
         n_tubes = 32.0;
        }

//    @component wue_aussen{                    /* WueStrahlungUndKonvektionRohr */
//          int i;
//          for (i=0; i<ND; i++)
//             {
//              t_fluid[i] = 525.00 + 273.15;
//              t_wall[i]  = 700.0;
//             }
//          l       = 27.180;
//          da      = 25.0e-03;
//          alpha   = 57.43;
//          epsilon = 0.80;
//          n_tubes = 32;
//          f       = 8.265;
//         }

   @component wue_aussen{                    /* WueKonvektion */
         int i;
         for (i=0; i<ND; i++)
            {
             t_fluid[i] = 264.50 + 273.15;
             t_wall[i]  = 600.0;
            }
         alpha   = 48.01;
         A       = 1215.38;
         f       = 1.0;
        }

   @component rg{
         int i;
         for (i=0; i<ND; i++)
            {
             T[i]     =  264.50 + 273.15;
            }
         T_in   = 264.50 + 273.15;
         T_out  = 188.36 + 273.15;
         cp     = 1069.5;
         m_dot  = 20.40;
         aquer  = 7.924;          /* aus: m_dot=rho*A*c */
         p      = 1.020e+05;
         length = 72.8;
        }

   @connect{
            m_dot_rg         = rg.m_dot;
            m_dot_wasser_in  = wasser.m_dot_in;
            m_dot_wasser_out = wasser.m_dot_out;

            t_rg_in          = rg.T_in;
            p_rg_in          = rg.p;
            rho_rg_in        = rg.rho              = 0.4609;
            h_wasser_in      = wasser.h_in;
            rho_wasser_in    = wasser.rho_in;

            t_rg_out         = rg.T_out;
            p_rg_out         = rg.p;
            rho_rg_out       = rg.rho              = 0.4609;
            h_wasser_out     = wasser.h_out;
```

```
            p_wasser_out     = wasser.p_out;

//             wue_innen.c      = wasser.c[0];

            wasser.T[i]       = wue_innen.t_fluid[i];
            wand.T2[i]       = wue_innen.t_wall[i];

/*            wand.T1[i]        = wue_aussen.t_wall[i]; */            /* Gleichstrom */

            wand.T1[0]       = wue_aussen.t_wall[5];                /* Gegenstrom */
            wand.T1[1]       = wue_aussen.t_wall[4];                /* Gegenstrom */
            wand.T1[2]       = wue_aussen.t_wall[3];                /* Gegenstrom */
            wand.T1[3]       = wue_aussen.t_wall[2];                /* Gegenstrom */
            wand.T1[4]       = wue_aussen.t_wall[1];                /* Gegenstrom */
            wand.T1[5]       = wue_aussen.t_wall[0];                /* Gegenstrom */

            rg.T[i]          = wue_aussen.t_fluid[i];



            wasser.Q_dot[i]     = wue_innen.dQ[i];
            wand.dQ2[i]       = wue_innen.dQ[i];

/*            wand.dQ1[i]       = wue_aussen.dQ[i]; */                /* Gleichstrom */

            wand.dQ1[0]       = wue_aussen.dQ[5];                     /* Gegenstrom */
            wand.dQ1[1]       = wue_aussen.dQ[4];                     /* Gegenstrom */
            wand.dQ1[2]       = wue_aussen.dQ[3];                     /* Gegenstrom */
            wand.dQ1[3]       = wue_aussen.dQ[2];                     /* Gegenstrom */
            wand.dQ1[4]       = wue_aussen.dQ[1];                     /* Gegenstrom */
            wand.dQ1[5]       = wue_aussen.dQ[0];                     /* Gegenstrom */

            rg.Q_dot[i]        = wue_aussen.dQ[i];
           }


@end
```

B.7.3 „Verdampferrohr.smi“

```
#include "RgWasserWAT.smd"
#include "Wue.smd"
#include "Wand.smd"
#include "Wasserbilanz.smd"
#include "RgEnergiebilanz.smd"

#include "arraydefs.h"

@interface Verdampferrohr : RgWasserWAT {

   @protected
      WasserbilanzVerdampferrohr*            dampf;
       WueKonvektion*                        wue_innen;
//        WueVerdampfungRohr*                   wue_innen;
      ThinWallTubes*                         wand;
//        WueStrahlungUndKonvektionRohr*        wue_aussen;
      WueKonvektion*                         wue_aussen;
      RgEnergiebilanz*                       rg;

   @export
```

```
      double delta_t                    [eq, doc: "Zeitschrittweite", unit: "s", min: 0.0];
      double zeit                       [eq, hist, doc: "Zeit", unit: "s", min: 0.0];
      double Q_dot_wasser               [eq, doc: "Wasserwaermeleistung", unit: "W"];
      double Q_dot_rg                   [eq, doc: "Rauchgaswaermeleistung", unit: "W"];
      double cp                         [doc: "spezif. Waermekapazitaet Rg", unit: "J/kg/K"];
     }

   @init {
          m_dot_wasser_in  = 2.5116;
          m_dot_wasser_out = 2.5116;
          cp               = 1100.1;
          zeit             = 0.0;
         }

@end

@implementation Verdampferrohr

   @eq cont zeit (time)
      {return time;}

   @eq cont delta_t (zeit)
      {return zeit-zeit[|last|];}

   @eq cont Q_dot_wasser (m_dot_wasser_in, m_dot_wasser_out, h_wasser_in, h_wasser_out)
      {return m_dot_wasser_out*h_wasser_out-m_dot_wasser_in*h_wasser_in;}

   @eq cont Q_dot_rg (m_dot_rg, cp, t_rg_in, t_rg_out)
      {return m_dot_rg*cp*(t_rg_in-t_rg_out);}

   @component dampf{ /* Initialisierungen in 'Wasserbilanz.smi' */ }

//    @component wue_innen{int i;              /* WueVerdampfungRohr */
//                  for (i=0; i<ND; i++)
//                     {
//                      t_wall[i]  = 540.0;           /* geschaetzt */
//                      t_fluid[i] = 252.62 + 273.15;
//                      dQ[i]      = 0.0;
//                      alpha[i]   = 8595.61;
//                     }
//                  ntubes = 32;
//                  p      = ;
//                  di     = 19.8e-03;
//                  l      = ;
//                  f      = ;
//                 }

   @component wue_innen{                               /* WueKonvektion */
                  int i;
                  for (i=0; i<ND; i++)
                    {
                     t_fluid[i] = 252.62 + 273.15;
                     t_wall[i]  = 540.0;
                    }
                  alpha   = 8600.0;
                  A       = 1430.60;
                  f       = 1.0;
                    }

   @component wand{int i;                              /* ThinWallTubes */
                 for (i=0; i<ND; i++)
                    {
                     T[i]  = 540.0;
                     T1[i] = 540.0;
```

```
                        T2[i] = 540.0;
                       }
                    rho     = 8000.0;
                    cp      = 477.0;
                    dm      = 22.4e-03;
                    b       = 2.6e-03;
                    l       = 86.45;
                    n_tubes = 32;
                   }

//    @component wue_aussen{int i;                  /* WueStrahlungUndKonvektionRohr */
//                    for (i=0; i<ND; i++)
//                       {
//                        t_wall[i] = 540.0;
//                        t_fluid[i]= ;
//                        dQ[i]     = 0.0;
//                       }
//                    alpha  = 53.0;
//                    epsilon= 0.80;
//                    f      = ;
//                    ntubes = 32;
//                    da     = 25.0e-03;
//                    l      = 86.45;
//                    }

   @component wue_aussen{                  /* WueKonvektion */
                      int i;
                      for (i=0; i<ND; i++)
                         {
                          t_fluid[i] = 457.49 + 273.15;
                          t_wall[i]  = 540.0;
                         }
                      alpha   = 53.0;
                      A       = 1430.60;
                      f       = 1.0;
                     }

   @component rg{   int i;
                    for (i=0; i<ND; i++)
                       {
                        T[i]     = 457.49 + 273.15;
                        Q_dot[i] = 0.0;
                       }
                    T_in   = 457.49 + 273.15;
                    T_out  = 264.50 + 273.15;
                    cp     = 1100.1;
                    m_dot  = 20.40;
                    aquer  = 0.08032;           /* aus: m_dot=rho*A*c */
                    p      = 1.0e+05;
                    length = 1.285;
               }

   @connect{
         m_dot_rg         = rg.m_dot;
         m_dot_wasser_in  = dampf.m_dot_in;
         m_dot_wasser_out = dampf.m_dot_out;

         t_rg_in          = rg.T_in;
         p_rg_in          = rg.p;
         rho_rg_in        = rg.rho             = 0.556;
         h_wasser_in      = dampf.h_in;
         rho_wasser_in    = dampf.rho_in;

         t_rg_out         = rg.T_out;
```

```
            p_rg_out          = rg.p;
            rho_rg_out        = rg.rho             = 0.556;
            h_wasser_out      = dampf.h_out;
            p_wasser_out      = dampf.p_out;

           dampf.T[i]          = wue_innen.t_fluid[i];
//           wand.T[i][ND_WL_1] = wue_innen.t_wall[i];
           wand.T2[i]           =wue_innen.t_wall[i];

//           wand.T[i][0]       = wue_aussen.t_wand[i];         /* Gleichstrom */

           wand.T1[0]          = wue_aussen.t_wall[5];             /* Gegenstrom */
           wand.T1[1]          = wue_aussen.t_wall[4];             /* Gegenstrom */
           wand.T1[2]          = wue_aussen.t_wall[3];             /* Gegenstrom */
           wand.T1[3]          = wue_aussen.t_wall[2];             /* Gegenstrom */
           wand.T1[4]          = wue_aussen.t_wall[1];             /* Gegenstrom */
           wand.T1[5]          = wue_aussen.t_wall[0];             /* Gegenstrom */

           rg.T[i]            = wue_aussen.t_fluid[i];

           dampf.Q_dot[i]      = wue_innen.dQ[i];
           wand.dQ2[i]         = wue_innen.dQ[i];

//            wand.dQ1[i]       = wue_aussen.dQ[i];      /* Gleichstrom */

           wand.dQ1[0]        = wue_aussen.dQ[5];          /* Gegenstrom */
           wand.dQ1[1]        = wue_aussen.dQ[4];          /* Gegenstrom */
           wand.dQ1[2]        = wue_aussen.dQ[3];          /* Gegenstrom */
           wand.dQ1[3]        = wue_aussen.dQ[2];          /* Gegenstrom */
           wand.dQ1[4]        = wue_aussen.dQ[1];          /* Gegenstrom */
           wand.dQ1[5]        = wue_aussen.dQ[0];          /* Gegenstrom */

           rg.Q_dot[i]        = wue_aussen.dQ[i];

           }

@end
```

B.7.4 „Ueberhitzerrohr.smi“

```
#include "RgWasserWAT.smd"
#include "Wasserbilanz.smd"
#include "RgEnergiebilanz.smd"
#include "Wue.smd"
#include "Wand.smd"

#include "arraydefs.h"
#include "zeitkonstanten.h"

@interface Ueberhitzerrohr : RgWasserWAT{

   @protected
      WasserbilanzUeberhitzerrohr*                dampf;
//      WueKonvRohrInnen*                           wue_innen;
      WueKonvektion*                              wue_innen;
//       WaermeleitungRohr*                          wand;
      ThinWallTubes*                              wand;
//       WueStrahlungUndKonvektionRohr*              wue_aussen;
      WueKonvektion*                              wue_aussen;
      RgEnergiebilanz*                            rg;
```

```
   @export
      double delta_t                    [eq, doc: "Zeitschrittweite", unit: "s", min: 0.0];
      double zeit                       [eq, hist, doc: "Zeit", unit: "s", min: 0.0];
      double Q_dot_wasser               [eq, doc: "Wasserwaermeleistung", unit: "W"];
      double Q_dot_rg                   [eq, doc: "Rauchgaswaermeleistung", unit: "W"];
      double cp                         [doc: "spezif. Waermekapazitaet Rg", unit: "J/kg/K"];
   }

   @init{
      m_dot_wasser_in  = 2.5116;
      m_dot_wasser_out = 2.5116;
      cp               = 1131.1;
      zeit             = 0.0;
     }

@end

@implementation Ueberhitzerrohr

   @eq cont zeit (time)
      {return time;}

   @eq cont delta_t (zeit)
      {return zeit-zeit[|last|];}

   @eq cont Q_dot_wasser (m_dot_wasser_in, m_dot_wasser_out, h_wasser_in, h_wasser_out)
      {return m_dot_wasser_out*h_wasser_out-m_dot_wasser_in*h_wasser_in;}

   @eq cont Q_dot_rg (m_dot_rg, cp, t_rg_in, t_rg_out)
      {return m_dot_rg*cp*(t_rg_in-t_rg_out);}

   @component dampf{ /* Initialisierungen in 'Wasserbilanz.smi' */ }

//    @component wue_innen{                    /* WueKonvRohrInnen */
//          int i;
//          for (i=0; i<ND; i++)
//             {
//           t_fluid[i] = 251.48 + 273.15;
//              t_wall[i]  = 700.0;
//             }
//          di      = 19.8e-03;
//          da      = 25.0e-03;
//          l       = 27.180;
//          rho     = 20.50;
//          Pr      = 1.2;
//          eta     = 10.0e-06;
//          lambda  = 20.0;
//          c       = 12.44;
//          n_tubes = 32;
//          xi      = 0.02;
//         }

   @component wue_innen{                       /* WueKonvektion */
         int i;
         for (i=0; i<ND; i++)
            {
             t_fluid[i] = 251.48 + 273.15;
             t_wall[i]  = 700.0;
            }
         alpha   = 1086.15;
         A       = 447.14         ;
         f       = 1.0;
        }
```

```
//    @component wand{                  /* Waermeleitung */
//          int i;
//          int j;
//          for (i=0; i<ND_WL; i++)
//             {
//           for (j=0; j<ND; j++)
//                {T[j][i] = 700.0;
//                 T1[j]   = 700.0;
//                 T2[j]   = 700.0;}
//             }
//          lambda  = 15.0;
//          rho     = 8000.0;
//          c_p     = 477.0;
//          di      = 19.8e-03;
//          da      = 25.0e-03;
//          l       = 27.180;
//          f       = 1.0;
//          n_tubes = 32;
//         }

   @component wand{                  /* ThinWallTubes */
         int j;
         for (j=0; j<ND; j++)
            {T[j]    = 700.0;
             T1[j]   = 700.0;
             T2[j]   = 700.0;}
         rho     = 8000.0;
         cp      = 477.0;
         dm      = 22.4e-03;
         b       = 2.6e-03;
         l       = 27.180;
         n_tubes = 32;
        }

//    @component wue_aussen{                  /* WueStrahlungUndKonvektionRohr */
//          int i;
//          for (i=0; i<ND; i++)
//             {
//              t_fluid[i] = 525.00 + 273.15;
//              t_wall[i]  = 700.0;
//             }
//          l       = 27.180;
//          da      = 25.0e-03;
//          alpha   = 57.43;
//          epsilon = 0.80;
//          n_tubes = 32;
//          f       = 8.265;
//         }

   @component wue_aussen{                  /* WueKonvektion */
         int i;
         for (i=0; i<ND; i++)
            {
             t_fluid[i] = 525.00 + 273.15;
             t_wall[i]  = 700.0;
            }
         alpha   = 57.34;
         A       = 447.14;
         f       = 1.0;
        }

   @component rg{
         int i;
         for (i=0; i<ND; i++)
```

```
          {
           T[i]     = 525.00 + 273.15;
          }
       T_in   = 525.00 + 273.15;
       T_out  = 457.49 + 273.15;
       cp     = 1131.1;
       m_dot  = 20.40;
       aquer  = 7.918;          /* aus: m_dot=rho*A*c */
       p      = 1.020e+05;
       length = 0.625;
      }

  @connect{
          m_dot_rg          = rg.m_dot;
          m_dot_wasser_in  = dampf.m_dot_in;
          m_dot_wasser_out = dampf.m_dot_out;

          t_rg_in          = rg.T_in;
          p_rg_in          = rg.p;
          rho_rg_in        = rg.rho              = 0.4609;
          h_wasser_in      = dampf.h_in;
          rho_wasser_in    = dampf.rho_in;

          t_rg_out         = rg.T_out;
          p_rg_out         = rg.p;
          rho_rg_out       = rg.rho              = 0.4609;
          h_wasser_out     = dampf.h_out;
          p_wasser_out     = dampf.p_out;

//             wue_innen.c      = dampf.c[0];

          dampf.T[i]       = wue_innen.t_fluid[i];
          wand.T2[i]       = wue_innen.t_wall[i];

/*           wand.T1[i]        = wue_aussen.t_wall[i]; */            /* Gleichstrom */

          wand.T1[0]       = wue_aussen.t_wall[5];                 /* Gegenstrom */
          wand.T1[1]       = wue_aussen.t_wall[4];                 /* Gegenstrom */
          wand.T1[2]       = wue_aussen.t_wall[3];                 /* Gegenstrom */
          wand.T1[3]       = wue_aussen.t_wall[2];                 /* Gegenstrom */
          wand.T1[4]       = wue_aussen.t_wall[1];                 /* Gegenstrom */
          wand.T1[5]       = wue_aussen.t_wall[0];                 /* Gegenstrom */

          rg.T[i]          = wue_aussen.t_fluid[i];




          dampf.Q_dot[i]     = wue_innen.dQ[i];
          wand.dQ2[i]       = wue_innen.dQ[i];

/*           wand.dQ1[i]       = wue_aussen.dQ[i]; */                /* Gleichstrom */

          wand.dQ1[0]       = wue_aussen.dQ[5];                     /* Gegenstrom */
          wand.dQ1[1]       = wue_aussen.dQ[4];                     /* Gegenstrom */
          wand.dQ1[2]       = wue_aussen.dQ[3];                     /* Gegenstrom */
          wand.dQ1[3]       = wue_aussen.dQ[2];                     /* Gegenstrom */
          wand.dQ1[4]       = wue_aussen.dQ[1];                     /* Gegenstrom */
          wand.dQ1[5]       = wue_aussen.dQ[0];                     /* Gegenstrom */

          rg.Q_dot[i]        = wue_aussen.dQ[i];
         }
```

```
@end
```

B.8 Einspritzkühler

B.8.1 „Einspritzkuehler.smi"

Die mathematische Modellierung des Einspritzkühlers steht in Kapitel 4.1.7 ab Seite 35.

```
#include <steam.h>

@interface Einspritzkuehler_cuts : SmileObject{

   @export
      double m_dot_in  [doc: "Dampfeintrittsmassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
      double m_dot_out [doc: "Dampfaustrittsmassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
      double m_dot_w   [doc: "Einspritzwassermassenstrom", unit: "kg/s", min: 0.0,
                         max: 1.0e+06];
      double h_in       [doc: "Dampfeintrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double h_out      [doc: "Dampfaustrittsenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double h_w        [doc: "Einspritzwasserenthalpie", unit: "J/kg", min: 0.0,
                         max: 1.0e+07];
      double p          [doc: "Druck", unit: "Pa", min: 0.0, max: 1000.0e+05];
      }

@end

@implementation Einspritzkuehler_cuts

@end




@interface EinspritzkuehlerOhneSpeicher : Einspritzkuehler_cuts{

   @export
      double m_dot_in [eq, doc: "Dampfeintrittsmassenstrom", unit: "kg/s", min: 0.0,
                       max: 1.0e+06];
      double h_out    [eq, doc: "Dampfaustrittsenthalpie", unit: "J/kg", min: 0.0,
                       max: 1.0e+07];
      double t_in     [eq, doc: "Dampfeintrittstemperatur", unit: "K", min: 0.0,
                       max: 5000.0];
      double t_out    [eq, doc: "Dampfaustrittstemperatur", unit: "K", min: 0.0,
                       max: 5000.0];
      double t_w      [eq, doc: "Einspritzwassertemperatur", unit: "K", min: 0.0,
                       max: 5000.0];
   }

   @init{
      m_dot_in  = 10.0;
      m_dot_out = 11.0;
      m_dot_w   = 1.0;
      h_in      = 3200.0e+03;
      h_out     = 2963.6e+03;
      h_w       = 600.0e+03;
      p         = 10.0e+05;
```

```
      t_in      = 643.2;
      t_out     = 643.2;
      t_w       = 415.6;
   }

@end

@implementation EinspritzkuehlerOhneSpeicher

   @eq cont m_dot_in (m_dot_out, m_dot_w)
      {return m_dot_out-m_dot_w;}

   @eq cont h_out (m_dot_in, m_dot_out, m_dot_w, h_in, h_w)
      {return m_dot_in/m_dot_out*h_in+m_dot_w/m_dot_out*h_w;}

   @eq cont t_out (p, h_out)
      {double T_schaetz   = 250.0+273.15;
       double rho_schaetz = 4.0;
       double tol_p       = 1.0e-07;
       double tol_h       = 1.0e-07;
       double vek[14];

       wph (p, h_out, T_schaetz, rho_schaetz, tol_p, tol_h, vek);

       return vek[IDX_T];}

   @eq cont t_in (p, h_in)
      {double T_schaetz   = 250.0+273.15;
       double rho_schaetz = 4.0;
       double tol_p       = 1.0e-07;
       double tol_h       = 1.0e-07;
       double vek[14];

       wph (p, h_in, T_schaetz, rho_schaetz, tol_p, tol_h, vek);

       return vek[IDX_T];}

   @eq cont t_w (p, h_w)
      {double T_schaetz   = 250.0+273.15;
       double rho_schaetz = 4.0;
       double tol_p       = 1.0e-07;
       double tol_h       = 1.0e-07;
       double vek[14];

       wph (p, h_w, T_schaetz, rho_schaetz, tol_p, tol_h, vek);

       return vek[IDX_T];}

@end
```

B.8.2 „Drossel.smi“

Die mathematische Modellierung der Drossel steht in Kapitel 4.1.2 ab Seite 25.

```
@interface Drossel : SmileObject{

   @export
      double m_dot_1     [eq, doc: "Massenstrom 1", unit: "kg/s"];
      double m_dot_2     [eq, doc: "Massenstrom 2", unit: "kg/s"];
      double p_1          [doc: "Druck 1", unit: "Pa"];
```

```
      double p_2          [doc: "Druck 2", unit: "Pa"];
      double h_1          [doc: "spezifische Enthalpie 1", unit: "J/kg"];
      double h_2          [eq, doc: "spezifische Enthalpie 2", unit: "J/kg"];

   @protected
      double k             [doc: "Faktor", unit: "kg/s/Pa^0.5"];
      double delta_p      [eq, doc: "Druckdifferenz", unit: "Pa"];
   }

   @init{
         k        = 1.0;
         p_1      = 1.1e+05;
         p_2      = 1.0e+05;
         h_1      = 1000.0e+03;
        }

@end

@implementation Drossel

   @eq cont h_2 (h_1, time)
      {return h_1;}

   @eq cont delta_p (p_1, p_2, time)
      {return p_1-p_2;}

   @eq cont m_dot_1 (delta_p, k)
      {
         if (delta_p > 1.0) {
            return k * sqrt (delta_p);
         } else if (delta_p < -1.0) {
            return -k * sqrt (-delta_p);
         } else {
            return k * (-0.25 * delta_p * delta_p + 1.25) * delta_p;
         }
      }

   @eq cont m_dot_2 (m_dot_1)
      {return m_dot_1;}

@end
```

B.9 Sonstige Smile-Dateien

B.9.1 „Integrator.smi“

Die mathematische Modellierung des Integrators ist in Kapitel 5.3 ab Seite 43 zu finden.

```
@interface Integrator : SmileObject{

   @export
      double x_ein           [doc: "Eintrittsgroesse"];
      double x_aus           [eq, doc: "Austrittsgroesse"];

   @protected
      double T_int           [doc: "Integrationszeitkonstante", unit: "s", min: 0.0];
   }

   @init{T_int = 0.1;}
```

```
@end

@implementation Integrator

   @eq diff x_aus (T_int, x_ein, x_aus)
      {return 1/T_int*(x_ein-x_aus);}

@end
```