Werner Arnus, BSc.

Simulation einer technischen Anlage aus der Domäne der Thermodynamik mit Hilfe von OpenModelica

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Telematik

eingereicht an der

Technischen Universität Graz

Betreuer:

Univ.-Prof.Dipl-Ing.Dr.techn. Franz Wotawa

Institut für Softwaretechnologie

Graz, März 2015

Kurzbeschreibung

Das Ziel dieser Masterarbeit ist die Simulation einer technischen Anlage. Für die Simulation wird die objektorientierte Beschreibungssprache Modelica und als Entwicklungsumgebung die Open-Source Version OpenModelica verwendet. Die physikalischen Grundlagen dieses Systems sind Großteils im Bereich der Thermodynamik und Strömungslehre enthalten. Die diversen Komponenten die zum Aufbau eines solchen Systems verwendet werden müssen, wurden physikalisch analysiert und einzeln in Modelica umgesetzt und simuliert. Die Komponenten wurden zu einem gesamten System verbunden um der Aufgabenstellung gerecht zu werden. Das fertige System erlaubt eine Analyse des Verhaltens für verschiedene Betriebspunkte welche am Ende dieser Arbeit exemplarisch gezeigt werden. Diese Arbeit zeigt die praktischen Einsatzmöglichkeiten von OpenModelica zur Simulation und Fehlersuche für reale technische Systeme. Das fertige System lässt sich leicht anpassen um auch auf zukünftige Änderungen vorbereitet zu sein.

abstract

The aim of this thesis is the simulation of a technical system. For the simulation, the object oriented language Modelica and as a development environment, the open source version of OpenModelica is used. The key areas of the physical foundations of this system are thermodynamics and fluid mechanics. The various components that must be used to build such a system were analyzed physically and individually implemented in Modelica and simulated. The components were connected to form the complete system, which allows analysis of the behavior for different operating points which are exemplarily shown at the end of this work. This work shows the practical application possibilities of OpenModelica to simulate real technical systems. The finished system can be adapted to be prepared for future changes.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Danksagung

Diese Diplomarbeit wurde im Studienjahr 2014/2015 am Institut für Softwareentwicklung an der Technischen Universität Graz durchgeführt.

An dieser Stelle mochte ich mich ganz herzlich bei Herrn Univ.-Prof. Dipl.-Ing. Dr. Franz Wotawa für die Bereitschaft bedanken, die Betreuung und Korrektur meiner Masterarbeit zu übernehmen.

Wichtig waren mir während meines Studiums meine Familie und im Besonderen meine Frau Julia, bei denen ich mich besonders bedanken möchte, da sie mich immer unterstützt, bestärkt und zu jeder Zeit an mich geglaubt haben.

Ebenfalls bedanke ich mich bei meinen Studienkollegen, mit denen ich eine erinnerungswürdige Zeit verbracht habe und die mich stets immer motiviert haben.

Inhalt

Abbildungsverzeichnis

Codeverzeichnis

1 Aufgabenstellung

Das Ziel dieser Masterarbeit ist die Entwicklung eines Modells eines technischen Systems aus dem Bereich der Thermodynamik. Der Fokus liegt auf der Verwendung von OpenModelica als Modellierungswerkzeug einer praktischen Anwendung. Dazu muss das System mit Hilfe gängiger Daten zu realen Gerätschaften konfiguriert werden können, wie zum Beispiel einem Datenblatt einer Pumpe.

Für die Simulation des Modells wird die objektorientierte Beschreibungssprache Modelica und als Entwicklungsumgebung die Open-Source Version OpenModelica verwendet. Die physikalischen Grundlagen dieses System sind großteils im Bereich der Thermodynamik und Strömungslehre enthalten. Die Thermodynamik ist ein physikalisches Teilgebiet, welches beschreibt, wie durch die Umverteilung verschiedener Aggregatzustände, Arbeit verrichtet wird. Die Strömungslehre behandelt das Verhalten von Flüssigkeiten und Gasen.

Die diversen Komponenten die zum Aufbau eines solchen Systems verwendet werden müssen, wurden physikalisch analysiert und einzeln in OpenModelica umgesetzt und simuliert. Danach wurden die Komponenten zu einem gesamten System verbunden um der Aufgabenstellung gerecht zu werden. Das fertige System erlaubt eine Analyse des Verhaltens für verschiedene Betriebspunkte welche am Ende dieser Arbeit exemplarisch gezeigt werden.

Der Hintergrund des Systems ist die Energierückgewinnung durch Verwendung von Kondensat und heißer Abluft. Ein technischer Prozess verbraucht Dampf und liefert als Abfallprodukte heiße Abluft und kaltes Kondensat. Um Energie beim Erstellen des Dampfs zu sparen, kann das alte Kondensat durch die heiße Abluft erhitzt und wieder als Dampf zurückgeführt werden.

Dieser Arbeit liegt ein reales System und dessen Auslegungsprogramm zur Grunde, welches leider für die Veröffentlichung nicht zur Verfügung steht. Die Ergebnisse dieser Arbeit wurden mit den Werten aus diesem Programm verglichen und validiert. Diese Werte dienten als Grundlage um ein erstes stabiles System zu entwickeln und auf diesem aufbauend ein allgemeines Modell des Systems zu entwickeln. Die Intention dieser Arbeit ist es ein allgemeines Modell des Systems zu entwickeln um das Auslegungsprogramm bei dessen Schwächen zu ergänzen. Das Auslegungsprogramm ist ein statisches Tool und dient dazu stabile Betriebspunkte zu analysieren, ist aber nur bedingt in der Lage Spezialfälle, wie das Anfahren des Systems, abzubilden. Hier soll diese Arbeit aushelfen und eine Analyse des Systems ermöglichen. Durch die daraus gewonnen Erkenntnisse kann das System noch weiter optimiert und Störfälle schneller behoben werden.

1.1 Aufbau und Struktur

Die Aufgabenstellung zeigte die Problemstellung und die Zielsetzung der Arbeit auf. Der Hauptteil ist in drei Themenbereiche unterteilt: Zuerst gibt es einen Überblick über Modelica und OpenModelica im Speziellen. Danach wird ein Überblick über die physikalischen Grundlagen und eine Beschreibung der technischen Elemente gegeben. Anschließend beschreibt diese Arbeit die Umsetzung der physikalischen Modelle in OpenModelica. Abschließend werden einige Ergebnisse präsentiert, die das Verhalten der Elemente und des gesamten Systems erläutern. Am Ende der Arbeit gibt es eine Zusammenfassung der daraus gewonnen Erkenntnisse.

2 Modelica

Modelica ist eine objektorientierte Programmiersprache zur Beschreibung physikalischer Modelle. Sie eignet sich zur Simulation von großen und komplexen Systemen und erlaubt multi-domain Modellierung [EMO99]. Multi-domain Modellierung erlaubt es Elemente, von verschiedenen physikalischen Domänen miteinander zu verbinden.

2.1 Ein kurzes Beispiel

Betrachten wir nun ein einfaches Modelica Beispiel anhand des Modells einer Mondlandefähre [Fri04, Kapitel 3.4]. Das Codefragment 2.1 zeigt das Modell der Mondlandefähre. Die Deklaration einer Modelica Klasse beginnt mit einem Schlüsselwort wie hier *class* oder *model*. Danach werden die Variablen definiert sowie optional ein Startwert zugewiesen. Mit dem *parameter*-Präfix werden Variablen definiert, die einmal zugewiesen, keinen anderen Wert für die Dauer der Simulation mehr annehmen.

Die Mondlandungsfähre hat eine Masse, Geschwindigkeit und Beschleunigung. Die Gravitation des Mondes hat einen Einfluss auf die Beschleunigung und somit auf die Flughöhe. Der Schub der Fähre wirkt der Gravitation entgegen und verringert die Beschleunigung. Die Masse der Fähre verringert sich durch den Verbrauch von Treibstoff, siehe Gleichung 2.1 und Zeile 11 im Code-Fragment 2.1. Die drei Differentialgleichungen erster Ordnung in 2.2 beschreiben die Zusammenhänge zwischen Beschleunigung, Geschwindigkeit und Flughöhe und sind in den Zeilen 12 bis 14 implementiert.

$$\frac{\delta Masse}{\delta t} = -Massenverlustrate * |Antriebsschub| \tag{2.1}$$

$$\frac{\delta Geschwindigkeit}{\delta t} = Beschleunigung \tag{2.2}$$

$$Beschleunigung = \frac{Schub - Masse * g}{Masse} \tag{2.3}$$

$$\frac{\delta Flughoehe}{\delta t} = Geschwindigkeit \tag{2.4}$$

Mittels *der()* werden Ableitungen nach der Zeit definiert, somit entspricht **der(x) = y** der Differentialgleichung $\frac{\delta x}{dt} = y$.

```
1  model Rocket
2      parameter String name;
3      Real mass(start = 1038.358);
4      Real altitude(start = 59404);
5      Real velocity(start = -2003);
6      Real acceleration;
7      Real thrust;
8      Real gravity;
9      parameter Real massLossRate = 0.000277;
10   equation
11     (thrust - mass * gravity) / mass = acceleration;
12     der(mass) = -massLossRate * abs(thrust);
13     der(altitude) = velocity;
14     der(velocity) = acceleration;
15   end Rocket;
```

Code 2.1: Beispiel Mondlandungsfähre.

Um eine Mondlandung zu simulieren muss noch eine Klasse *MoonLanding* entworfen werden, die die Höhe der Gravitationsanziehung auf das Gefährt berechnet. Diese Kraft ist eine Funktion aus der Gravitation des Mondes, dessen Masse und Radius sowie der Entfernung der Landefähre zum Mond. Beispiel 2.3 zeigt wie so eine Steuerung aussehen könnte. In Zeile 8 wird eine Instanz vom oben definierten Typ Rocket mit dem Namen Apollo12 erstellt. Die Eigenschaften des Mondes werden

durch die Klasse *CelestialBody* abgedeckt (Alg. 2.2). Der Schub der Landefähre soll zu den definierten Zeitpunkten *thrustEndTime* und *thrustDecreaseTime* geändert werden. Dies lässt sich einfach durch einen *If*-Block verwirklichen, indem man auf die eingebaute *time*-Variable zurückgreift.

```
1  model CelestialBody
2      constant Real g = 0.00000000006672;
3      parameter Real radius;
4      parameter String name;
5      Real mass;
6    end CelestialBody;
```

Code 2.2: Einfaches Modell eines Himmelskörpers.

Das Beispiel zeigt auch wie Zugriffsbeschränkungen in Modelica verwendet werden. In Zeile 4 bewirkt das Schlüsselwort *protected*, dass alle nachfolgenden Variablen nur für Code in dieser Klasse, oder davon abgeleiteten Klassen, sichtbar sind. Im Gegensatz dazu erlaubt *public*, dass jeder Code, der eine Instanz von dieser Klasse hat, die Variable sehen kann. In Modelica sind alle Variablen standardmäßig *public*, somit sind in unserem Beispiel *force1* und *force2* auch *public*.

```
1  model MoonLanding
2    parameter Real force1 = 36350;
3    parameter Real force2 = 1308;
4    Rocket apollo(name = "apollo12");
5    CelestialBody moon(name = "moon",
6                       mass = 7.382e+22, radius = 1736000.0);
7  protected
8    parameter Real thrustEndTime = 210;
9    parameter Real thrustDecreaseTime = 43.2;
10 equation
11   apollo.thrust = if time < thrustDecreaseTime then force1
12                   else if time < thrustEndTime then force2
13                   else 0;
14   apollo.gravity = moon.g * moon.mass /
15                   (apollo.altitude + moon.radius) ^ 2;
16 end MoonLanding;
```

Code 2.3: Beispiel für die Simualtion der Landung.

In den Abbildungen 2.1 und 2.2 sieht man den Verlauf der Flughöhe

und des Schubes. Simuliert wurde vom Zeitpunkt t=0 bis t=230 Sekunden. Der Schub der Fähre ist zuerst sehr hoch und wird bei t=43.2 Sekunden stark reduziert und bei t=210 Sekunden ganz ausgeschaltet. In der Abbildung 2.1 sieht man, dass die Fähre zum Zeitpunkt t=208 Sekunden die Höhe 0, und somit die Mondoberfläche, erreicht hat.

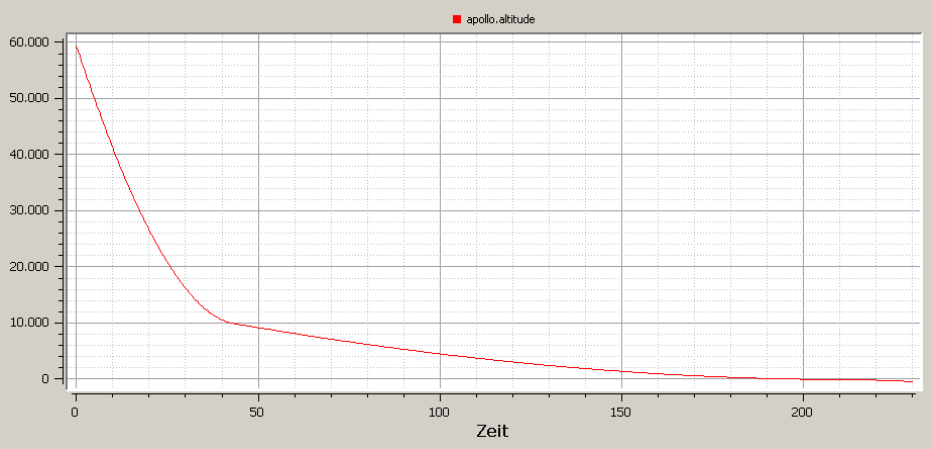


Abbildung 2.1: Die Flughöhe der Landefähre.

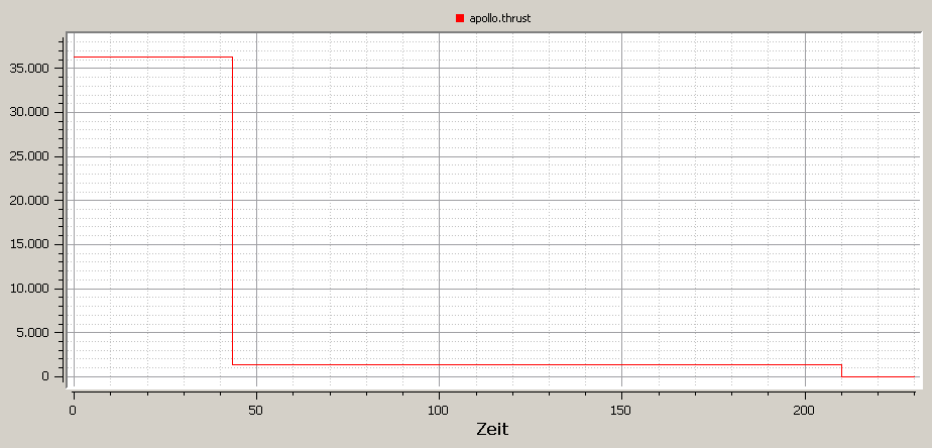


Abbildung 2.2: Der Schub der Landefähre.

2.2 Modelica Geschichte

Im September 1996 hatte sich eine Gruppe von Designern zusammen getan. Ihr Ziel war es eine neue einheitliche objekt-orientierte Modellierungssprache zu entwickeln. Der erste Artikel über Modelica erschien 1997 [SHJ97].

Anfangs wurde die Gruppe als *Technical Comittee 1* in EuroSim und als *Technical Chapter on Modelica* in der *Society for Computer Simulation International* gegründet. Im Jahr 2000 wurde dann die heute bestehende *Modelica Association* als eine internationale unabhängige non-profit Organisation gegründet, die sich das Ziel gesetzt hat die Modelica Sprache zu entwickeln und verbreiten. Dabei wurde Modelica nicht entwickelt um vorhandene starke Domain-spezifische Beschreibungsmittel wie VHDL-AMS abzulösen, sondern den Zugang für "Modellierern aus den Bereichen der Mechanik, Verfahrenstechnik, Automatisierungstechnik etc. ein Beschreibungs- und Simulationsmittel für heterogene Systeme" [SCHS00] zu liefern.

Heutzutage verwenden viele Firmen und Universitäten Modelica: z.B. ([mod14]) :

- Firmen aus der Automobilindustrie wie zum Beispiel BMW,Audi,Ford
- ABB
- EDF
- Siemens

Es gibt verschiedene Versionen von Modelica, darunter auch einige Open-Source Varianten. Eine kurze Auswahl an Modelica Versionen sind:

- OpenModelica
- JModelica
- Dymola
- Wolfram SystemModeler

Im Verbundprojekt GENSIM wird auf Basis von Modelica ein generisches Simulationswerkzeug MOSILBA entwickelt [NG05]. Insgesamt sechs Fraunhofer-Institute arbeiten daran, ein Simulationswerkzeug zur

Simulation komplexer heterogener technischer Systeme zu entwickeln. MOSILAB erlaubt die Unterstützung von variablen Modellstrukturen zur Simulationslaufzeit.

Um OpenModelica Modelle auch in einem Browser starten zu können hat Tom Short ein Programm entworfen um OpenModelica Modelle zu JavaScript zu kompilieren [Sho15]. Dadurch können OpenModellica Modelle mit JavaScript zur Verfügung gestellt werden. Das Buch 'Modelica by Example' von Michael Thiller [Thi15] verwendet die so erstellten JavaScripte um eine bessere Einführung in OpenModelica anbieten zu können. So kann man interaktiv, online und ohne OpenModelica installieren zu müssen, die Beispiele im Buch ausprobieren. Das Buch wurde anfangs durch ein Kickstarter-Projekt finanziert. Mittlerweile findet man unter den Sponsoren namhafte Größen wie Wolfram Research, Maplesoft und Siemens. Die HTML Version ist und soll gratis bleiben, weitere Versionen wie als PDF oder gebundene Ausgabe sind noch nicht verfügbar aber in Arbeit.

2.3 Besonderheiten

Im Gegensatz zu den meisten Programmiersprachen basiert Modelica auf Gleichungen und nicht auf Zuweisungen [FR03]. Gleichungen sind mächtiger als einfache Zuweisungen, da sie in alle Richtungen gültig sind. Ein einfaches Beispiel ist hier die Ohmsche Gleichung: $U = R * I$. Abhängig von den gegebenen Variablen muss man in den meisten gängigen Programmiersprachen diese Gleichung umschreiben [Fri04]. In Modelica wird diese einfache Gleichung in alle mögliche Zuweisungen auf gesplittet:

1. $U := R * I$
2. $R := \frac{U}{I}$
3. $I := \frac{U}{R}$

.

Modelica benützt all diese Zuweisungen und verwendet einen Lösungsalgorithmus um das Modell zu lösen.

Intern erzeugen die meisten Simulatoren ein System von Differentialen-Algebraischen Gleichungen welche numerisch gelöst werden. In Modelica wird ein komponentenorientierter Ansatz verfolgt, dadurch gleicht die Modellstruktur dem realen System stärker.

Gleichungen können in 4 Arten aufgeteilt werden:

Differentialgleichungen sind Gleichungen mit Ableitungen nach der Zeit.
algebraische Gleichungen enthalten keine Ableitungen.
partielle Differentialgleichung enthalten Ableitungen nach anderen Variablen als der Zeit: $\frac{\delta x}{\delta t} = \frac{\delta^2 x}{\delta z^2}$.
Differenz Gleichungen beschreiben den Zusammenhang zwischen Variablen verschiedener Zeitpunkte, zum Beispiel $x(t+1) = x(t) + 1$.

2.3.1 Kontinuierliche und zeit-diskrete Modelle

In Modellica es ist möglich Zeit-kontinuierliche und -diskrete Modelle miteinander zu verbinden. Dadurch kann man ein Modell mit Differentialgleichungen beschreiben und gleichzeitig diskrete Vorgänge beobachten. Zum Beispiel kann das Modell der Mondlandungsfähre von 2.1 mit einem Regler erweitert werden.

Das System ist zeit-kontinuierlich und kann mit zeit-diskreten Elementen, wie einem Regler, kombiniert werden [Fri04, Kapitel 13.2.5.4]. In Modelica kann man sehr einfach einen diskreten Regler simulieren. Dazu gibt es den *sample* Befehl, der angibt in welchen Zeitabständen Zeit-diskrete Werte aus einem kontinuierlichen System ausgelesen werden. Zusätzlich gibt es noch *discrete-time* Variable, diese sind stückweise konstante Signale, die ihren Wert nur zu bestimmten, durch den *event* Befehl ausgelösten, Zeitpunkten ändern. Sie haben die Besonderheit, dass ihre Ableitung nach der Zeit gleich null ist. Im Gegensatz dazu stehen zeit-kontinuierliche Variablen, die sich mit der Zeit kontinuierlich ändern.

Objektorientierte Modelle

In Modelica werden objektorientierte Modelle folgendermaßen aufgefasst: Objektorientiert wird als Struktur-Konzept wahrgenommen. Die mathematischen Modelle sind schichtweise aufgebaut und können wiederverwendet und miteinander kombiniert werden. Es wird auf drei Arten

strukturiert: hierarchisch, Komponenten-Verbindungen und Vererbung [Fri04, Chapter 2.2]. Die Modelle werden in Modelica durch deklarative Gleichungen beschrieben. Deklaratives Programmieren wird durch die Mathematik beeinflusst, wo es heißt, dass eine Aussage hält, anstatt eine genaue Schritt-für-Schritt Anweisung zu geben. Dadurch kann man Modelle einfach definieren, in dem man eine Differentialgleichung definiert und nicht manuell die einzelnen Zeitschritte aufarbeiten muss.

Durch den deklarativen und objektorientierten Weg von Modelica erlaubt es einen höheren Abstraktionsgrad zu erreichen. Einige Aktionen die in anderen Programmiersprachen nötig sind, wie zum Beispiel den Transport von Daten zwischen Objekten, muss man in Modelica nicht machen. Weiter unten werden Objekte vom Typ *connector* 2.3.2 erklärt. Diese erlauben es Objekte miteinander zu verbinden, da im Hintergrund automatisch die notwendigen zusätzlichen Gleichungen eingefügt werden.

2.3.2 Klassentypen

Die objekt-orientierte Herangehensweise führt dazu, dass ziemlich alles in Modelica eine Klasse ist. Um eine leichtere Handhabung und Wartung zu erreichen gibt es sieben spezielle vordefinierte Klassen: [Fri04]

Die *model* - Klasse ist der am meisten verwendete Klassentyp. Es entspricht fast genau der grundlegenden Klassendefinition von Modelica. Es gibt nur sehr geringe Einschränkungen.

Eine weitere Klasse ist vom Typ *record*. Diese Klassen dürfen keine Gleichungen enthalten sondern dienen nur zur Datenverwaltung.

Die *type* Klasse dient zum Definieren neuer Klassentypen wie zum Beispiel ***type*** *Matrix = Real[3,3]*.

Die *block* Klasse fügt zum allgemeinen Klassenverhalten noch eine kausale Datenfluss-Richtung hinzu. Es gibt *input* und *output* Präfixe die die Richtung angeben.

Der *function* Typ ist ein weiterer Spezialfall. Er repräsentiert mathematische Funktionen die unabhängig berechnet werden können und somit nicht mit der Außenwelt interagieren. Ähnlich wie der *block* Typ gibt es

auch hier *input* und *output* Präfixe. Es gibt weiter einige Einschränkungen: jede Variable muss entweder den *input* oder *output* Präfix enthalten. Anders als bei *model* gibt es keine Gleichungen nur Zuweisungen.

Der *package* Typ dient zur Verwaltung von Namensräumen und zur Ordnung der Modelle. Dieser Typ kann nur Deklarationen und andere Klassen enthalten. Dabei werden alle zugehörigen Klassen in der package-Datei gespeichert. Das Beispiel 2.4 zeigt anhand des bekannten Moonlande-Beispiels wie so eine Klasse aussehen kann, alle Klassen innerhalb können auf einander zugreifen.

```
package PackageExample
  model Rocket
    parameter String name;
    Real mass(start = 1038.358);
    Real altitude(start = 59404);
    Real velocity(start = -2003);
    Real acceleration;
    Real thrust;
    Real gravity;
    parameter Real massLossRate = 0.000277;
  equation
    (thrust - mass * gravity) / mass = acceleration;
    der(mass) = -massLossRate * abs(thrust);
    der(altitude) = velocity;
    der(velocity) = acceleration;
  end Rocket;

  model MoonLanding
    parameter Real force1 = 36350;
    parameter Real force2 = 1308;
  protected
    parameter Real thrustEndTime = 210;
    parameter Real thrustDecreaseTime = 43.2;
  public
    Rocket apollo(name = "apollo12");
    CelestialBody moon(name = "moon",
                mass = 7.382e+22, radius = 1736000.0);
  equation
    apollo.thrust = if time < thrustDecreaseTime then force1
                    else if time < thrustEndTime then force2
                    else 0;
    apollo.gravity = moon.g * moon.mass /
```

```
                    (apollo.altitude + moon.radius) ^ 2;
  end MoonLanding;

  model CelestialBody
    constant Real g = 0.00000000006672;
    parameter Real radius;
    parameter String name;
    Real mass;
  end CelestialBody;
end PackageExample;
```

Code 2.4: Ein Beispiel zur Verwendung der package Deklaration.

Die *connector* Klasse dient als eine Basisklasse für Anschlüsse zwischen den Elementen.

Connector

Ein Modelica *connector* definiert ein externes Interfaces für die Interaktion. Zwei oder mehrere Objekte können mit einem *connector* des gleichen Typs miteinander verbunden werden. In Modelica werden Verbindungen als Gleichungen verwirklicht. Es sind sowohl kausale als auch akausale Verbindungen möglich. Bei akausalen Verbindungen ist die Richtung des Datenflusses nicht vorher zu wissen.

Ein einfaches Beispiel ist der elektrische connector:

```
connector electricPort
  Voltage U;
  Current I;
end electricPort;
```

Code 2.5: Ein Modell für einen elektrischen connector.

Wie bei anderen Objekten in Modelica kann man auch bei *connector* eigene Typen definieren. Dadurch lassen sich eigene Verbindungen sehr einfach anlegen und verbinden. Ein *connector* für thermische Systeme kann so aussehen:

```
connector flowPort
  flow Real m;
```

```
  Real P;
  Real T;
  flow Real mV;
end flowPort;
```

Code 2.6: Ein Modell für einen thermischen connector.

Ein *connector* hat 2 Arten von Variablen:

- **flow** Variable repräsentieren fließende Einheiten.
- Normale Variable sind meist vom Typ Real.

flow Variablen

Über den Befehl *connect()* werden zwei Objekte miteinander verbunden. Dazu müssen beide Modelle über den selben connector-Typ verfügen. Ein connect verbindet die Anschlüsse zu einem gemeinsamen Knotenpunkt. Dieser hat mehrere Eigenschaften:

Gleichsetzen Alle Zustände des Connector's werden gleich gesetzt : $Pin1.U = Pin2.U$

Sum-to-Zero Die Summe aller *flow* Variablen wird auf Null gesetzt : $Pin1.i + Pin2.i = 0$.

Anhand dieser zusätzlichen Regeln lassen sich Objekte einfach verbinden.

Partielle Modelle

Für den hierachischen Aufbau der Modelle werden meist zuerst Basis-Klassen erschaffen [Fri04]. Ein einfaches Beispiel hierfür ist ein elektrischer Anschluss:

```
model electricPartial
  Port positive,negative;
  Voltage U;
  Current I;
equation
  U = positivePort.u - negativePort.u;
  I = positivePort.i;
```

```
  positivePort.i + negativePort.i = 0;
end electricPartial;
```

Code 2.7: Ein Modell für eine elektrische Basisklasse mit Strom und Spannung.

So ein einfaches partielles System kann man als ein Zwei-Anschluss-System bezeichnen. Die meisten elektrischen, und auch thermischen, Objekte können als Erweiterung eines solchen Systems mit zwei Anschlüssen angesehen werden. Durch die Verwendung dieser partiellen Objekte kann man sehr einfach eigene Objekte aufbauen. Die hierarchische Struktur erlaubt es auch unkompliziert Teile in jeder Abstraktionsebene auszutauschen.

2.3.3 Aufbau einer Klasse

Wichtige Definitionen in Modelica sind der *Algorithm* und *Equation* - Tag in einer Klasse.

Algorithm

Die Algorithm - Definition erlaubt es einer Klasse als einfache Funktion zu fungieren. Dabei werden die Gleichungen der Reihe nach abgearbeitet und haben einen definierten Input und Output.

Equation

Der equation-Tag definiert den Hauptteil einer Klasse. In diesem werden die Gleichungen geschrieben, die Reihenfolge in der diese stehen spielen keine Rolle. Wie schon oben in 2.3 beschrieben, werden alle Gleichungen aufgearbeitet.

2.4 OMEdit: Der OpenModelica Editor

In OpenModelica ist auch ein grafischer Editor inkludiert. Dieser verwendet Qt und benützt Corba um mit den OpenModelica Compiler zu

kommunizieren. Durch diesen Editor lassen sich einfach eigene Modelle zusammen stellen und auch komplexe Systeme erstellen. Der Editor erlaubt es auch eigene Symbole für Klassen zu erstellen [AT10].

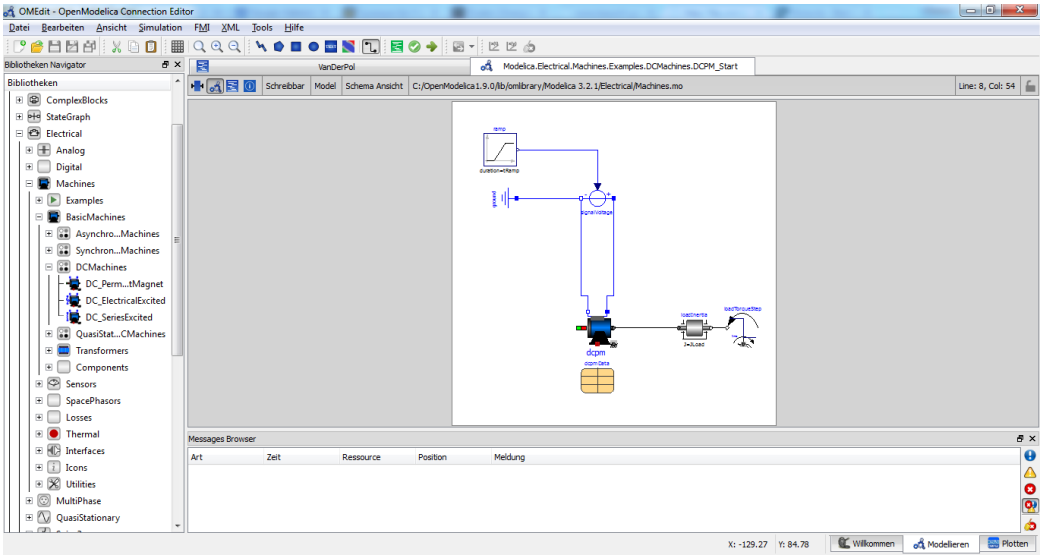


Abbildung 2.3: Der Open Modelica Connection Editor

Bild 2.3 zeigt den Editor. Auf der linken Seite befinden sich die Bibliothek mit den verfügbaren Klassen. Diese Komponenten können mittels Drag-and-Drop in die Schema-Ansicht in der Mitte gezogen werden. In den Modelica Bibliotheken sind viele Objekte verschiedener Domänen vorhanden.

Die grafische Oberfläche bietet auch die Möglichkeit, eigene Grafiken und Symbole für Modelle zu zeichnen. Mittels Drag-and-Drop werden Elemente angeordnet und auch verbunden (siehe den connect Befehl 2.3.2). Zusätzlich kann man auch in der grafischen Oberfläche die Parameter der Elemente eintragen. Als solche gelten als *parameter* definierte Variablen, die für die Dauer der gesamten Simulation gleich bleiben, wie zum Beispiel die Wärmeleitfähigkeit eines Objektes oder der Widerstand eines elektrischen Elementes. Zusätzlich ist es auch möglich Anfangszustände in der grafischen Oberfläche einzutragen. Somit können Personen ohne große Programmierkenntnisse ein technisches System zusammen stellen und simulieren. Der Editor visualisiert auch das Ergebnis

der Simulation, dabei kann neben dem normalen Verlauf über die Zeit x(t) auch der Verlauf einer Variable zu einer anderen Variable x(y) dargestellt werden.

3 Die physikalischen Grundlagen

Das vorherige Kapitel beschäftigt sich mit der Einführung in Modelica und gibt einen kurzen Überblick über dessen Eigenschaften. Nun werden die physikalischen Grundlagen beschrieben und eine kurze Beschreibung der einzelnen technischen Elemente wie Pumpen, Ventile und Separatoren gegeben.

3.1 Das System

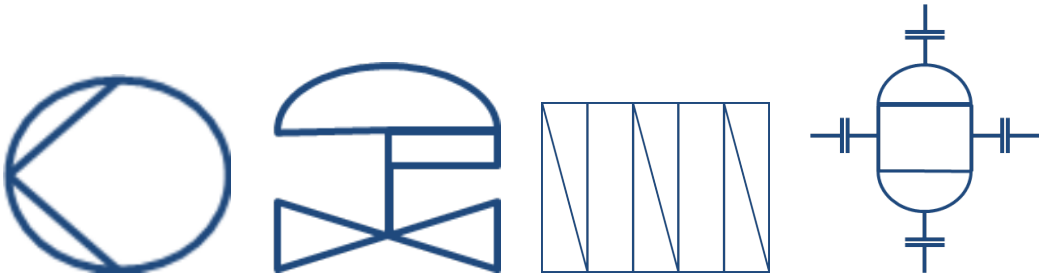
Es wird nun kurz das System und dessen Komponenten erklärt, danach werden die Komponenten einzeln und im Detail beschrieben. Abbildung 3.1 zeigt die Schaltzeichen der einzelnen Komponenten.

Das zu simulierende System besteht aus folgenden Komponenten:

- Einen Wärmetauscher der das flüssige Kondensat erhitzt.
- Einem Ventil zur Druckregulierung.
- Ein Separator als Behälter der den flüssigen Anteil vom gasförmigen trennt und als Speicher dient.
- Eine Pumpe um das flüssige Kondensat zu bewegen.

Der Wärmetauscher erhitzt das Kondensat mithilfe der zugeführten heißen Luft. Das erhitzte Kondensat wird dann durch ein Ventil in einen Separator geleitet. Das Ventil verringert den Druck und stellt somit die Bedingungen für das Verdampfen her. Würde Dampf bereits im Wärmetauscher entstehen, hätte dies eine Zerstörung des Systems zur Folge.

Daher wird durch das Ventil der Druck so geregelt, dass das Kondensat erst im Separator verdampfen kann. Im Separator befinden sich



(a) Pumpe. (b) Ventil. (c) Wärmetauscher. (d) Separator.

Abbildung 3.1: Die Schaltsymbole für die verschiedenen Elemente.

schlussendlich dann Dampf und Kondensat. Der Dampf wird über das obere Ende des Separators abgegeben. Das übrige Kondensat wird durch eine Umlaufpumpe wieder dem Wärmetauscher zugeführt und der Prozess

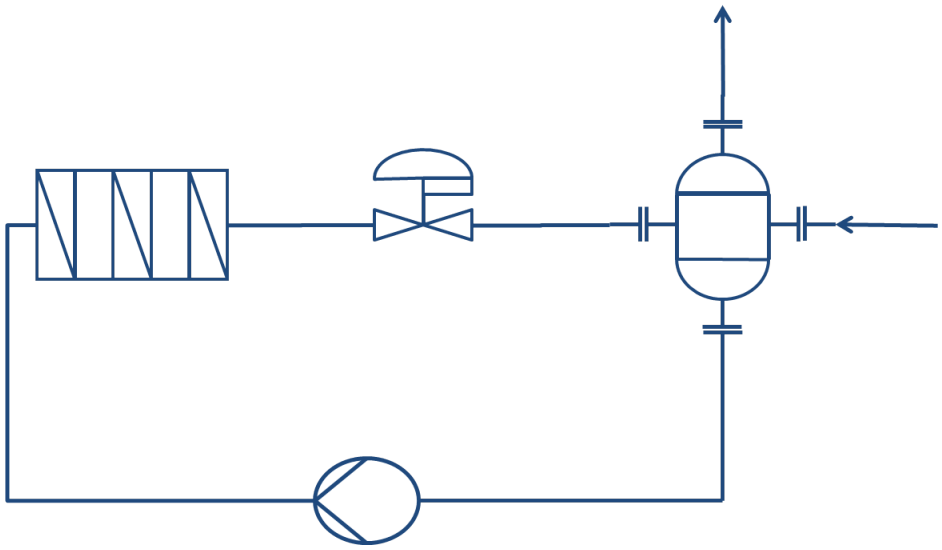


Abbildung 3.2: Das zu simulierende System: Von einem Separator wird das Medium durch eine Pumpe zu einem Wärmetauscher gepumpt. Das dort erhitzte Kondensat wird durch ein Ventil druck-geregelt und kommt dann wieder in den Separator wo sich der Dampf vom Kondensat trennt.

beginnt von vorne. Die Umlaufpumpe erhöht den Druck des Kondensats wieder erheblich, damit der Wärmetauscher dieses auch stark erhitzen kann.

Um das System zu stabilisieren werden Regler an der Umlaufpumpe, Wärmetauscher und Ventil benötigt. Das Bild 3.2 zeigt eine schematische Darstellung des Systems.

3.2 Die Basis-Formeln

Die Modellierung des Systems erfordert die genaue Betrachtung des Verhaltens von Wasser und des Wasserdampfes unter verschiedenen Umgebungsbedingungen. Druck und Temperatur beeinflussen sehr stark das Verhalten und bestimmen den Zustand des Mediums. Um die Eigenschaften des Wassers und Dampfes zu berechnen wurden die Formeln des [IAP07] verwendet.

Die *The International Association for the Properties of Water and Steam upshape (IAPWS)*ist ein internationaler Verband. Die IAPWS stellt Berechnungen für Wasser und Dampf zusammen und gibt Richtlinien zur Berechnung heraus. Im IAPWS Bericht von 2007 [IAP07] werden Formeln für fünf physikalische Bereiche der Zustände von Wasser angegeben. Diese Bereiche beschreiben das Verhalten von Wasser abhängig von Druck und der Temperatur und decken ein breites Gebiet ab:

- 273.15 K $\leq$ T $\leq$ 1073.15 K , für p $\leq$ 100 MPa.
- 1073.15 K $\leq$ T $\leq$ 2273.15 K , für p $\leq$ 50 MPa.

3.2.1 Einteilung der Geltungsbereiche

Bild 3.3 zeigt die Regionen. Die Bereiche lassen sich durch die zugrunde liegende fundamentalen Gleichungen 3.1 bis 3.4 beschreiben. Die Gleichungen sind entlang der Grenzübergänge zwischen den Bereichen, innerhalb gewisser Toleranzen, untereinander konsistent.

Die Bereiche 1 und 2 basieren auf der spezifischen Gibbs Energie $g(p,T)$, der 3. Bereich ist durch die Helmholtz Energie $f(\rho,T)$ definiert

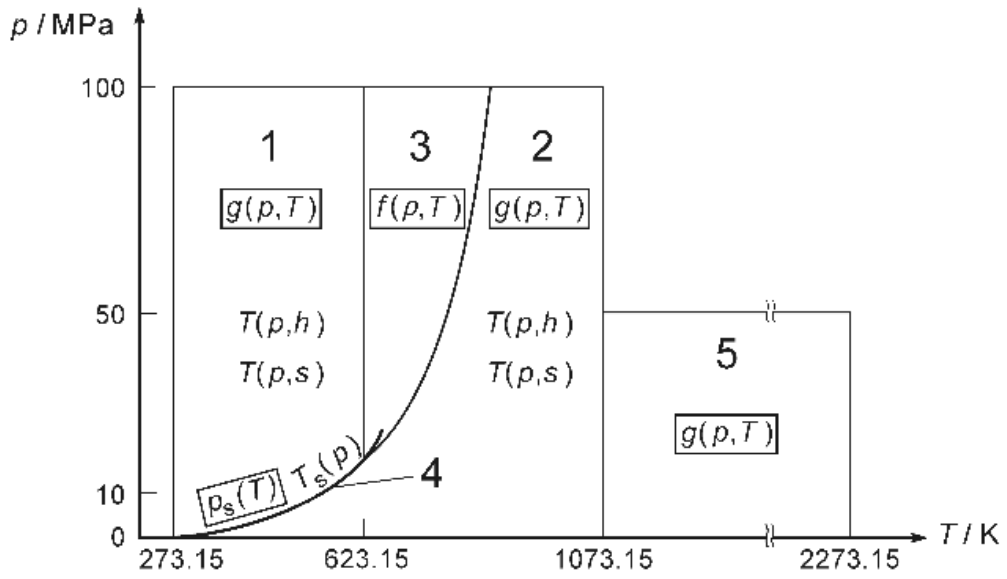


Fig. 1. Regions and equations of IAPWS-IF97.

Abbildung 3.3: Regionen von IAPWS-97 [IAP07]

und der 5. Bereich ist ein sehr heißer Bereich, auch definiert durch eine andere $g(p,T)$ Gleichung. Die Sättigungskurve entspricht dem 4. Bereich und ist durch eine Druck und Temperatur abhängige Gleichung definiert.

Die Gleichungen basieren meist auf der Gibbs Energie in der dimensionslosen Form $\gamma = \frac{g}{RT}$. Gleichung 3.1 beschreibt den 1. Bereich wobei n_i,I_i und J_i Koeffizienten sind. Die Basisgleichungen für den 2. Bereich sind in 2 Teile aufgespalten: einem idealen Gasanteil und einem residual Anteil, siehe Gleichungen 3.2. Der 3. Bereich wird durch die spezifische Helmholtz Energie in Gleichung 3.3 beschrieben. Der 4. Bereich bildet die Sättigungskurve ab und wird durch eine quadratische Gleichung berechnet. Der sehr heiße 5. Bereich wird wieder durch die Gibbs Energie beschrieben 3.4 und ist wie der zweite Bereich auch in zwei Teile aufgeteilt..

Alle thermodynamischen Eigenschaften können von diesen allgemeinen und dimensionslosen Gleichungen abgeleitet werden. Im Bericht [IAP07] wird auch auf die computergestützte Berechnung Rücksicht genommen

und für jeden Bereich Testwerte mit den Ergebnissen angegeben.

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \sum_{i=1}^{34} n_i (7,1-\pi)^{I_i} (\tau - 1,222)^{J_i} \tag{3.1}$$

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^o(\pi,\tau) + \gamma^r(\pi,\tau) \tag{3.2}$$

$$\gamma^o = ln\pi + \sum_{i=1}^{9} n_i^o * \tau_i^o$$

$$\gamma^r = \sum_{i=1}^{43} n_i \pi^{I_i} (\tau - 0.5)_i^J$$

$$\frac{f(\rho,T)}{RT} = \Phi(\delta,\tau) = n_1 ln\delta + \sum_{i=2}^{40} n_i \delta^{I_i} \tau^{J_i} \tag{3.3}$$

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^o(\pi,\tau) + \gamma^r(\pi,\tau) \tag{3.4}$$

$$\gamma^o = ln\pi + \sum_{i=1}^{6} n_i^o \tau^{J_i^o}$$

$$\gamma^r = \sum_{i=1}^{6} n_i \pi^{I_i} \tau^{J_i}$$

Kenngrössen

Die Kenngrössen um den Zustand von Wasser zu beschreiben sind:

- Spezifische Volumen $v[m^3/kg]$
- Spezifische Enthalpie $h[J/kg]$
- Spezifische Wärmekapazität $c_p[J/kgK]$
- Sättigungsdruck $p_s[bar]$

Das spezifische Volumen gibt das Verhältnis zwischen Volumen zu Masse an und ist somit der Kehrwert der Dichte. Die Enthalpie, $H[J]$, ist eine Maßeinheit für die Energie eines Systems. Die spezifische Enthalpie ist auf die Masse bezogen und ist eine intensive Größe im Gegensatz zur Enthalpie. Die spezifische Wärmekapazität ist auch auf die Masse bezogen. Sie kann auch als Energiemenge aufgefasst werden, die nötig ist um ein Kilogramm um ein Kelvin zu erhitzen.

3.2.2 Enthalpie

Die Enthalpie ist ein Maß für die Energie eines Mediums. Sie wird auch Wärmeinhalt genannt. Das Zeichen für die Enthalpie sind ein H in *[J]* für die allgemeine Enthalpie und ein kleines h in *[J/kg]* für die spezifische Enthalpie. Diese spezifische Enthalpie bezieht sich auf die Masse des Mediums und ist somit eine intensive Grösse. [Luco5]

Gemäss des Energieerhaltungssatzes kann Energie nicht verloren gehen. Dies ist zur Berechnung der Verdampfung besonders wichtig. Da Energie nicht verloren gehen kann, kommt es zu keiner Änderung der Summe bei Phasenübergänge.

3.3 Pumpen

Die Leistung von Pumpen wird mittels der Förderhöhe angegeben. Die Förderhöhe gibt an, wie hoch eine Pumpe eine Flüssigkeit, z.B. Wasser, fördern kann. Da die Erdanziehung auf die Wassersäule wirkt, muss die Pumpe eine Gegenkraft aufwenden und diese ist proportional zur förderbaren Höhe. Da $g \approx 9.81 \frac{m}{s^2}$ ist, kann vereinfacht der geförderte Druck berechnet werden. Demnach entsprechen 10 Meter Förderhöhe ungefähr 1 bar Leistung der Pumpe. Die Leistung der Pumpe wird drehzahlgeregelt eingestellt.

3.3.1 Pumpen- und Anlagenkennlinie

Die Pumpenkennlinie beschreibt den Zusammenhang zwischen der förderbaren Höhe und dem Durchfluss. Unter Förderhöhe versteht man die

von der Pumpe nutzbare Arbeit. Meist ist sie in [m] angegeben und entspricht der förderbaren Wasserhöhe. Wenn der Druck steigt, sinkt der Durchfluss und vice versa. Bild 3.4 zeigt eine Kennlinie für verschiedene Drehzahlen.

Genauso beschreibt die Anlagenkennlinie den Zusammenhang zwischen Durchfluss und Druckverlust in einem System. Der Druckverlust fasst die Widerstände des gesamten Systems zusammen. In der Regel hat sie die Form einer Parabel. Das bedeutet, dass die Druckverluste in einer Anlage quadratisch mit dem Volumenstrom ansteigen:

$$X = dP / V^2 \tag{3.5}$$

Der Schnittpunkt der Anlagenkennlinie und der Pumpenkennlinie gibt den neuen Betriebspunkt der Pumpe an [BB10, Chapter 8.4].

3.3.2 Berechnung des neuen Durchflusses

Ausgehend von der alten Druckdifferenz, des alten Durchflusses und des aktuellen Gegendruckes, wird der neue Durchfluss berechnet. Wie in 3.5 zu sehen ist, ist der aktuelle Betriebspunkt der Schnittpunkt der Pumpenkennlinie und der Anlagenkennlinie. Wenn sich die aktuelle Anlagenkennlinie (in blau) verschiebt, durch einem neuen Gegendruck, entsteht eine neue Anlagenkennlinie (in orange). Dadurch ergibt sich ein neuer Schnittpunkt.

Wenn nun eine Änderung des Druckverlusts eintritt verschiebt sich die Anlagenkennlinie, zum Beispiel siehe Anlagenkennlinie 2. Bei verringertem Druckverlust kann mehr gefördert werden und umgekehrt. Um die neue Kennlinie zu berechnen wird bei gleich bleibenden Durchfluss der Druck verändert. Dadurch ergibt sich eine neue Parabel mit:

$$X2 = dP_{neu} / V_{alt}^2 \tag{3.6}$$

Der neue Betriebspunkt ist der Schnittpunkt der neuen Anlagenkennlinie mit der Pumpenkennlinie. Dort lässt sich der neue Durchfluss ablesen. Obwohl die Förderhöhe nicht dem neuen Druckverlust entspricht, ergibt sich ein neuer Durchfluss. Dieser Durchfluss bewirkt einen neuen

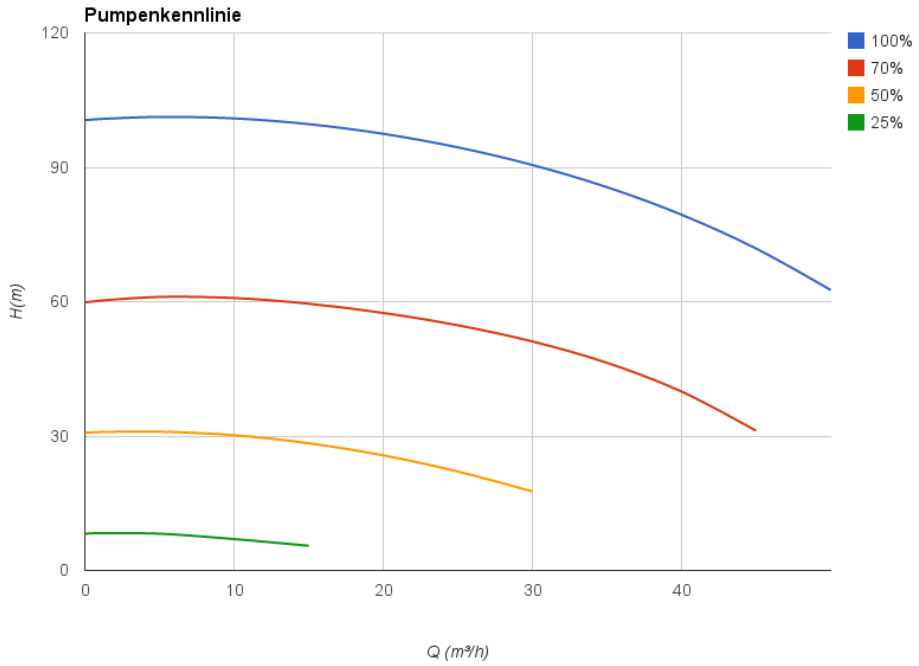


Abbildung 3.4: Die Pumpenkennlinie zeigt den Arbeitsbereich einer Pumpe an. Die Prozentangabe bezieht sich auf die maximale Leistung der Pumpe.

Druckverlust, der sich auf der Anlagenkennlinie befindet und dem neuen Betriebspunkt entspricht. Dieser neue Betriebspunkt gibt an wie viel die Pumpe nun fördert und welcher Durchfluss gezogen wird. [Küm07, Chapter 4.3.5]

3.4 Ventile

Ventile regeln den Durchfluss durch Leitungen und bewirken einen Druckabfall. Je weiter ein Ventil geschlossen wird, desto geringer wird die Durchflussöffnung. Je weiter ein Ventil den Durchfluss einschränkt desto höher ist der Druckabfall am Ventil. Zur Berechnung des Druckabfalls werden einige Kenngrössen benötigt, die nachfolgend erklärt werden.

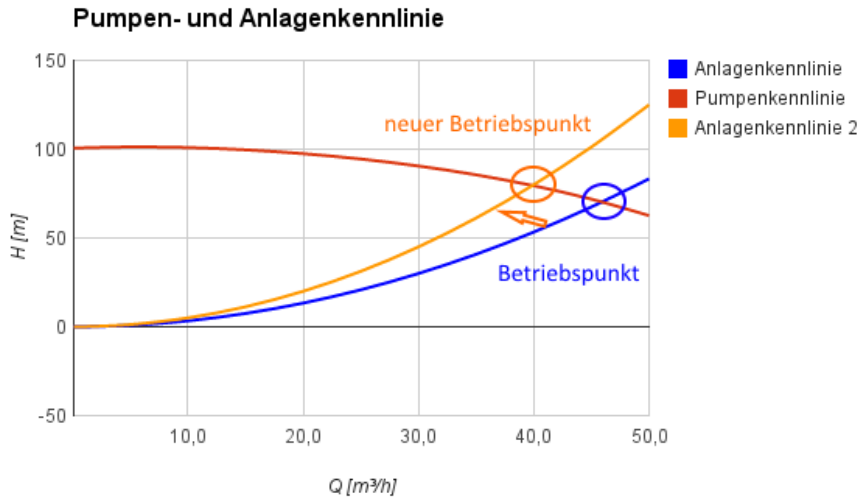


Abbildung 3.5: Die Anlagenkennlinie beschreibt die Auswirkungen eines Systems und dessen Druckverlustes.

3.4.1 Kenngrössen

Wichtige Kenngrössen zur Beschreibung von Ventilen sind [DS07]:

- K_V Wird Durchflussfaktor oder auch Durchflusskoeffizient genannt. Er gibt den Durchfluss $\dot{V}$ in $\frac{m^3}{h}$ für Wasser bei einer Temperatur von 5 bis 30 °C an, der bei einem Druckverlust von 1 bar auftritt.
- C_V Ist der K_V Wert auf das nicht metrische Mass-System umgerechnet : $C_V = K_V * 0.86$.
- K_{VS} Ist der wichtigste Wert eines Ventils. Es ist das Kennzeichen für Ventile , und gibt den K_V Wert für den Nennhub H_{100} an.
- H_{100} Bezeichnet den Hub bei dem das Ventil voll geöffnet ist.
- Durchmesser: Der Durchmesser des Eingangs und des Ausgangs.
- ζ Der Druckverlustbeiwert gibt in der Strömungslehre den Druckverlust durch einer Rohrleitung oder Armatur an.

Die inhärente Durchflusscharakteristik beschreibt den relativen Durchfluss abhängig vom Hub. Über diesen Zusammenhang kann der aktuelle

Durchfluss und damit der Druckverlust berechnet werden.

3.4.2 Berechnung der Druckdifferenz

$$K_V = \dot{V} * \sqrt{\frac{1bar}{\Delta p} \frac{\rho}{1000 \frac{kg}{m^3}}} \tag{3.7}$$

$$\zeta = \frac{\Delta p}{\frac{\rho}{2} * v^2} \tag{3.8}$$

Wie in der Gleichung 3.8 ersichtlich ist, bezieht sich der Druckverlustbeiwert ζ auf den Druckverlust Δp in einem Teilstück bei einer Geschwindigkeit [GC05]. Der Druckverlustbeiwert ζ kann bei nacheinander geschaltete Armaturen einfach addiert werden. Dabei ist der Druckverlustbeiwert immer auf einen bestimmten Querschnitt bezogen.

$$v = \frac{\dot{V}}{D^2} \tag{3.9}$$

Da der aktuelle Durchfluss $\dot{V}$ in $\frac{m^3}{h}$ gegeben ist, kann die Geschwindigkeit v durch die Amatur mit dem Durchmesser D berechnet werden(siehe 3.9).

$$\Delta p = \frac{\frac{\rho}{2} * v^2}{\zeta} \tag{3.10}$$

Durch Umformen der Gleichung 3.8 in Gleichung 3.10 kann der Druckverlust durch die Armatur berechnet werden.

Herleitung der ζ Werte

Der ζ Wert ist immer auf einen Durchfluss bezogen. Der Durchfluss durch das Ventil kann durch den K_V Wert berechnet werden. Die inhärente Flusscharakteristik ermöglich die Berechnung des aliquoten Anteils des Durchflusses in Abhängigkeit vom eingestellten Hub des Ventils [Hero4].

Vereinfachter Ansatz

Ein vereinfachter Ansatz bietet auch die direkte Berechnung des Druckverlustes über den K_V Wert. So kann die Gleichung 3.7 zur Gleichung 3.11 verallgemeinert werden. Dadurch reicht es den K_V Wert für den gegebenen Hub und damit den Druckverlust zu berechnen.

$$\Delta p = \frac{\dot{V}^{\,2}}{K_V} \tag{3.11}$$

3.4.3 Ventile und Temperatur

Das erhitzte Kondensat nach dem Wärmetauscher soll bis knapp unter die Verdampfungsgrenze erhitzt sein. Dies wird durch verschiedene Regler erreicht, dazu aber später mehr in 4.3.

Auf der Eingangsseite des Ventils ist das Kondensat auf knapp unter der Verdampfungstemperatur erhitzt. Durch den Druckabfall an dem Ventil kann der Sättingungsdruck unterschritten werden und das Kondensat verdampft. Ziel des Regelkreises soll es sein, ein Maximum des Dampfes zu erzeugen und dabei auf der Eingangsseite des Ventils das Kondensat sicher unter der Verdampfungsgrenze zu halten. Die Temperatur wird nicht durch das Ventil beeinflusst, aber wie schon erwähnt sinkt durch den Druckabfall auch die Sättigungstemperatur des Kondensates. Hierzu muss die Sättigungstemperatur für die aktuelle Ventilöffnung berechnet werden.

In 4.2.4 wird genauer auf die Bedeutung der Temperatur eingegangen.

3.5 Separator

3.5.1 Allgemeine Erklärung

Ein Separator wird verwendet um zwei Medien von einander zu trennen. In dem gegeben System wird er verwendet um Kondensat und Dampf zu trennen. Durch eine Öffnung an der Oberseite des Behälters kann der

Dampf abtransportiert werden. Weiter fungiert der Separator als Speichermedium. Kondensat wird von ihm entnommen und erhitzt wieder eingeführt. Das erhitzte Kondensat vermischt sich im Behälter mit dem alten, wodurch sich eine neue Temperatur einstellt. Der Separator hat vier Anschlüsse: Einer dient als Ausgang zur Pumpe, einer als Eingang vom Ventil, einer als Ausgang für den Dampf und ein weiterer Eingang in den Separator dient als Quelle neuen Kondensates. Das neu zugeführte Kondensat dient als Ersatz für den entnommenen Dampf.

3.5.2 Das Modell

Wie schon oben erwähnt ist ein Separator auch ein Speichermedium, dadurch ändern sich auch die Eigenschaften des gespeicherten Mediums mit der Zeit. Die wichtigsten Eigenschaften sind die Temperatur, der Druck, die gespeicherte Menge und die gasförmigen Dampfanteile. Im folgenden werden ihre Berechnung kurz erklärt.

Die Temperatur

Die Temperatur im Separator ist von den Zu- und Abflüssen abhängig. Die neue Temperatur lässt sich mittels der Massenströme , der Dichte und Wärmeleitfähigkeit der beteiligten Ströme berechnen.

Die Menge

Der Zulauf und Abfluss des Separators werden mittels Differentialgleichungen modelliert:

$$V + dV_in - dV_{out} = 0 \tag{3.12}$$

V gibt das gespeicherte Volumen, dV_in und dV_{out} den Zufluss bzw. den Abfluss an. Die Zuflüsse setzen sich aus den 2 Eingangsanschlüssen zusammen: der erste wird mit dem Ventil V1 verbunden und erhält das erhitzte Kondensat und den erzeugten Dampf. Am 2. Anschluss hängt die Versorgung mit neuem Kondensat um den Verlust durch Dampf auszugleichen, um ein gleichbleibendes stabiles Fülllevel des Separators zu gewährleisten.

Es gibt zwei Abflüsse : der erste ist der Anschluss für die Rückführung des gewonnen Dampfes. Der zweite stellt den Kreislauf mit der Pumpe her. Über diesen Ausfluss zieht die Pumpe eine gewisse Menge an Kondensat. Ein spezieller Fall ist das Anfahren des Systems, da zu diesem Zeitpunkt die Pumpe noch nicht eingeschaltet ist. So hängt der Ausfluss aus dem Separator vom Druck und der Menge des gespeicherten Kondensates ab. Der Druck im Behälter muss gleich dem Außendruck am Austrittspunkt sein und die Geschwindigkeit kann durch Umformen der Gleichung 3.13 zur Gleichung 3.14; unter der Bedingung, dass der Strömungsquerschnitt des Abflusses viel kleiner als der des Behälters ist; berechnet werden. $(p - p_a)$ und $(z - z_a)$ sind die Druckdifferenz zwischen dem Außendruck und dem Druck im Behälter und die Höhendifferenz zwischen dem Austrittspunkt und der Füllhöhe des Behälters. Der neue Massestrom ergibt sich durch die Größe der Austrittsöffnung, siehe 3.15 [vB04, Kapitel 4.2.3]

$$p + \frac{1}{2} * \rho * c^2 + g * \rho * z = p_a + \frac{1}{2} * \rho * c_a^2 + g * \rho * z_a \tag{3.13}$$

$$c_a = \sqrt{\frac{2 * (p - p_a)}{\rho} + 2 * g * (z - z_a)} \tag{3.14}$$

$$\dot{m} = \rho * c * A \tag{3.15}$$

Gas-Anteile

Da es das Ziel dieses Wärmerückgewinnungssystems ist, erhitzten Dampf zu produzieren, wird der gasförmige Anteil der in den Separator strömt, auch als Ausgangsgrösse verwendet. Der Ausgang des gesamten Systems hängt am oberen Ausgang des Separators, durch welchen der gesamte gasförmige Anteil entweicht. Dadurch speichert der Separator nur flüssiges Kondensat.

Druck

Der Separator kann als ein Druckpotential mit Öffnung zur Aussenwelt angesehen werden. Dadurch wir der Druck im Separator als quasi kon-

stant angenommen. Da der Separator auch einen Anschluss an die restliche Maschine hat, ist dies eine gute Näherung.

3.6 Wärmetauscher

Ein Wärmetauscher transferiert Energie in Form von Wärme von einem Medium zu einem anderen. In dem vorliegenden Versuchsaufbau wird Wärme von dem zurück geführten Dampf auf Kondensat im Regelkreislauf übertragen.

Dabei wird von einem Strom die Energie ohne direkten Kontakt, über eine Trennwand , an einen 2. Strom transferiert.

Bei dem vorliegenden Versuchsaufbau behandeln wir einen Gas zu Flüssig Wärmetauscher.

3.6.1 Formen von Wärmetauscher

Man kann WT in drei Hauptgruppen anhand des geometrischen Aufbaues einteilen: Gleichstrom,Gegenstrom und Kreustrom Wärmetauscher. Beim Gleichstromwärmetauscher laufen die 2 Ströme nebeneinander, bei der Gegenstrom Variante fliessen die Ströme in entgegensetzte Richtungen, bei 2 nebeneinander liegenden Rohren. Bei der Kreuzstromvariante sind die Rohre in einem Kreuz angeordnet.

Eine weitere Unterscheidung kann man hinsichtlich der Art der Wärmeübertragung vornehmen: direkte, indirekte, halb-direkte Wärmeübertragung.

3.6.2 Wärmetauscher in Modellbildung

Beim 9. Simulations Symposium 1997 wurde ein Beispiel für eine Wärmetauschersimulation in Modelica vorgestellt [Mat97] . Dabei handelte es sich um einen Flüssig zu F̈lussig Wärmetauscher.

4 Umsetzung

Nach der Beschreibung der Grundlagen im Bereich der technischen Systeme und Modelica als Umgebung, widmet sich dieses Kapitel der Umsetzung. Zuerst wird das gesamte System noch einmal beschrieben und die Zielsetzung noch einmal erläutert. Danach folgt eine genauere Beschreibung der einzelnen Komponenten und ihre Interaktion miteinander.

4.1 System

Im Bild 4.1 ist das gesamte System schematisch abgebildet, es besteht aus einer Zirkulationspumpe die Kondensat fördert. Von der Zirkulationspumpe kommt das Kondensat in einen Wärmetauscher, der das Kondensat erhitzt. Das erhitzte Kondensat wird dann durch ein Ventil befördert. Durch das Ventil kommt es zu einem Druckverlust wodurch die Sättigungstemperatur unter die aktuelle Temperatur des Kondensates sinkt. Hinter dem Ventil befindet sich ein Separator, in dem das Kondensat verdampft. Der gasförmige Anteil wird abgegeben und die Differenz mit neuem Kondensat aufgefüllt. Das neue Kondensatgemisch kommt wieder zur Zirkulationspumpe, wodurch sich der Kreis schließt.

4.1.1 Bedingungen

Das Ziel des Systems ist es ein Maximum an Dampf zu erzeugen. Dabei hängen viele Faktoren miteinander zusammen.

Zum einen wird mittels eines Wärmetauschers das Kondensat erhitzt. Je weiter das Kondensat erhitzt ist, desto mehr Druck muss durch die

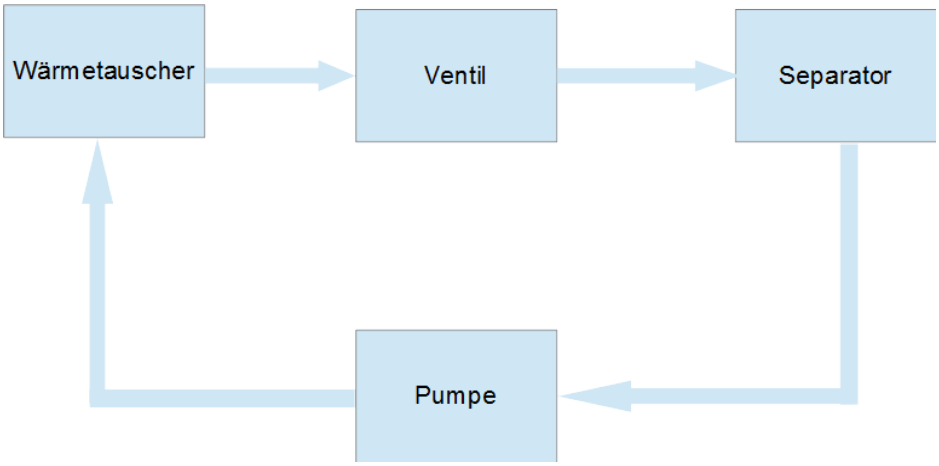


Abbildung 4.1: Das System besteht aus einer Pumpe die das Kondensat durch einen Wärmetauscher zu einem Ventil fördert und einem Ventil, dass den Druck verringert und so das Kondensat zum Verdampfen bringt und in einem Separator endet.

Pumpe erzeugt werden. Sonst kommt es zur Verdampfung in den Leitungen und dem Wärmetauscher, siehe 3.6. Dies muss unter allen Umständen verhindert werden.

Das Ventil soll den Druck optimal senken, damit ein Maximum an Dampf entsteht. Die Sättigungstemperatur ist abhängig von Druck und Temperatur des Kondensates.

Durch das Ventil kommt es zu einem Druckverlust, wodurch das Kondensat nach dem Ventil verdampfen kann. Weiters kommt es durch den Druckverlust auch zu einem Rückdruck im System. Dieser Druck verursacht entsprechend der Anlagenkennlinie 3.3 einen neuen Durchfluss. Durch diesen neuen Fluss kommt es zu einem neuen Druck und auch die geförderte Menge an Kondensat ändert sich. Um diese Auswirkungen zu kompensieren muss die Drehzahl der Pumpe verändert werden.

4.2 Grund-Modelle

Das gesamte System besteht aus vielen einzelnen Komponenten welche nun genauer beschrieben werden. Zuerst wird der *connector* vorgestellt der alle Komponenten miteinander verbindet. Danach wird die Basisklasse eines Elementes mit zwei Anschlüssen vorgestellt. Zum Abschluss wird auf die einzelnen Komponenten genauer eingegangen und auf ihre jeweiligen Besonderheiten hingewiesen.

4.2.1 Fluss

Ähnlich dem 2009 vorgestellten *stream*-connector [Mod09] wurde ein eigener connector entwickelt. Da zusätzlich zu dem normalen füssigen Massestrom auch ein Massestrom für den gasförmigen Anteil benötigt wurde.

Der verwendete *connector* 2.3.2 wird als Fluss bezeichnet. In diesem sind alle relevanten Eigenschaften des Fördermediums angegeben. Diese sind:

flow m Der Massestrom des flüssigen Anteils in [kg/h]
P Der Druck in [bar]
T Die Temperatur in [C]
flow mV Der Massestrom des gasförmigen Anteils in [kg/h]

Über den Druck und die Temperatur lassen sich die Eigenschaften des Fördermediums berechnen. Ein Beispiel dafür ist die Berechnung der Sättigungstemperatur bei einem gegebenen Druck. Da das Fördermedium im System in der Regel nicht verdampfen soll, ist es wichtig die Temperatur unter der Sättigungstemperatur zu halten. Dies wird mittels Reglern erreicht 4.3. Die Flussrichtung wird immer als positiv angenommen: der Massestrom in ein Element ist positiv und der Massestrom aus einem Element ist negativ. Wie schon im Modelica-Teil erklärt 2.3.2, werden bei einem *connector* durch eine flow - Variable der Anschluss eines Elementes zu einem Knotenpunkt mit den geltenden Regeln. Damit ist sichergestellt, dass die Summe aller zufließenden Masseströme gleich den abfließenden Masseströme ist.

4.2.2 Basisklasse

Die Basisklasse für viele Elemente ist ein einfaches Modell eines Elementes mit zwei Anschlüssen. In diesem Modell sind die grundlegenden Verknüpfungen zwischen dem Eingang und dem Ausgang realisiert. Im Grunde entspricht so eine Basisklasse dem connect-Befehl eines Connectors. Zusätzlich werden noch Hilfsvariablen eingeführt um eine Änderung zuzulassen. Ein Beispiel hierfür ist die Wärmeänderung im Wärmetauscher: $outletPort.T = inputPort.T + dT$. Dadurch können in einer abgeleiteten Klasse einfach die Änderungen berechnet und dann zugewiesen werden.

Es gibt zwei Arten von Basisklassen: die Standard-Klasse erlaubt nur eine Änderung der Temperatur und des Druckes, aber nicht die der Durchflüsse. Diese Art ist für die meisten Elemente ausreichend. Das Codefragment 4.1 zeigt diese Basisklasse mit derer man den Druck und die Temperatur am Ausgang beeinflussen kann. Ein spezieller Fall ist das Ventil, bei dem es zu einer neuen Berechnung des gasförmigen Anteils kommt. Dazu später mehr 4.2.4.

```
1  partial model TwoPort
2    Models.StreamConnector inputPort;
3    Models.StreamConnector outletPort;
4  protected
5    Real dT(start = 0);
6    Real dP(start = 0);
7  equation
8    outletPort.T = inputPort.T + dT;
9    outletPort.P = inputPort.P + dP;
10   outletPort.mV + inputPort.mV = 0;
11   outletPort.m + inputPort.m = 0;
12 end TwoPort;
```

Code 4.1: Die Basisklasse für fast alle Modelle.

4.2.3 Zirkulationspumpe

Die Berechnung des neuen Durchflusses der Zirkulationspumpe erfolgt wie in Abschnitt 3.3.2 beschrieben. Hierzu gibt es eine Basisklasse *Pumpe* welche folgende Elemente hält :

Vnew ist der neue Volumenstrom.
Pnew ist die neue Druckdifferenz.
rpmPercentage bestimmt die Leistung der Pumpe in [%] der maximalen Leistung.

Eine Pumpe hat folgende 3 Anschlüsse:

inputPort Representiert den Eingang der Pumpe.
outputPort Representiert den Ausgang der Pumpe.
pressurePort Ist der Anschluss für den Rückdruck durch das Ventil.

Wie in 3.3 beschrieben, fördert die Pumpe den Fluss. Dabei kommt es zu einer Druckerhöhung und Durchflussänderung. Das Modell der Pumpe beeinflusst den Druck am Ausgang durch erhöhen des dP der Basisklasse wie oben beschrieben. Durch andere Elemente im Kreislauf kann es zu einem Gegendruck kommen. So kommt es zum Beispiel durch ein Ventil oder Rohrverengung zu einem Engpass und dadurch sinkt der Durchfluss und der Druck steigt. Dieser Gegendruck wirkt auf die Pumpe und muss berücksichtigt werden. Dazu wird der durch das Ventil verursachte Gegendruck durch einen eigenen Connector mit der Pumpe verbunden. Der Druckverlust am Ventil wird als negativer Wert an die Pumpe übermittelt , damit dort der neue Betriebspunkt berechnet werden kann.

Parameter

Folgende Parameter werden für eine Pumpe benötigt und müssen verknüpft werden:

- rpmPercentage
- mDrawn

Beispiel

Betrachtet man ein einfaches System mit einer Pumpe und einem Ventil, so müssen sowohl die Pumpe als auch das Ventil gesteuert werden. Als Beispiel wird ein System angenommen, für das folgende Parameter eingestellt werden:

startPumpHub Die Drehzahl in% zu Beginn der Simulation.
startValveHub Die Ventilöffnung in % der maximalen Öffnung.
targetP Der Druck den die Pumpe liefern soll.

4.2.4 Ventil

Wie bei Pumpen wird eine Basisklasse *Valve* verwendet. Diese Klasse definiert das Verhalten eines Ventils im Allgemeinen.

Für die Simulation eines spezifischen Ventils müssen noch einige Parameter definiert werden.

Cv Der Cv Wert wie oben (3.4.1) beschrieben.
D Der Durchmesser des Ventils in [m].
rho Die Dichte des fluiden Materials wird mit ρ in $[kg/m^3]$.

Das Ventil wird durch Angabe des Hubs in % angesteuert. Dadurch wird das Ventil entlang der Ventilkurve geschlossen und geöffnet. Mittels eines Ventilreglers wird der Durchfluss so geregelt, wie später in 4.3.3 beschrieben wird.

Das Ventil hat einen weiteren Anschluss , mittels dessen der Rückdruck durch das Ventil zur Pumpe übermittlet wird (siehe 4.2.3).

Berechnung des Dampf-Anteiles

Wie in 3.4 beschreiben senkt das Ventil den Druck. Durch diesen Abfall sinkt auch die Sättingungstemperatur des Kondensates. Dadurch kann nach dem Ventil das Kondensat verdampfen und es muss im Modell der Anteil an flüssigen und gasförmigen Anteilen neu berechnet werden. Die Berechnung des Verhältnisses vom flüssigen zum gasförmigen Anteil des Kondensates wird mit Hilfe der Enthalpie berechnet.

$$h_{in} = X * h_{liquOut} + h_{vapOut} * (1 - X) \tag{4.1}$$

Mit X ist der flüssige Anteil am gesamten, in den Eingang kommenden, Kondensates gemeint. Somit ist die resultierende Menge an Dampf gleich dem Eingang multipliziert mit $1 - X$. Die Enthalpie wird wie schon

erwähnt durch die Gleichungen des IAFPS kalkuliert 3.2. Zur Berechnung der Enthalpien wird der aktuelle Druck vor und nach dem Ventil genommen. Der Druck nach dem Ventil ist der Eingangsdruck minus des Druckabfalles, verursacht durch das Ventil. Die Formeln des 3.2 erlauben es zusätzlich auch die Enthalpien des flüssigen sowie gasförmigen Anteils bei Sättigung zu berechnen 3.2.2. Weiter kann auch der Zustand des Mediums bei gegebenen Druck und Temperatur berechnet werden, dabei werden die Regionen wie in Grafik 3.3 angegeben beachtet.

Das Verhältnis X lässt sich durch die Enthalpien berechnen. Mit h_{in} ist die Enthalpie des eingehenden Massestromes gemeint. Zur Berechnung dieser Enthalpie wird der aktuelle Druck am Eingang des Ventils und die aktuelle Temperatur benötigt. Analog dazu kann die Enthalpie auch für den Ausgang berechnet werden. Mit h_{vapOut} ist die Enthalpie des gasförmigen Anteils bei einem gewissen Ausgangsdruck bei Sättigung gemeint. Bei $h_{liquOut}$ handelt es sich um den flüssigen Anteil.

Die Formel 4.1 lässt sich umformen um X zu berechnen:

$$X = \frac{(h_{in} - h_{vapOut})}{(h_{liquOut} - h_{vapOut})} \tag{4.2}$$

$$mV = m_{in} * (1 - X) \tag{4.3}$$

4.2.5 Wärmetauscher

Wie schon oben in 3.6 verwendet ein Wärmetauscher ein Medium auf der primären Seite um ein anders Medium auf der sekundären Seite zu erhitzen. Durch Wärmeübertragung im Wärmetauscher wird ein Temperaturausgleich zwischen den beiden Medien angestrengt. Um den Wärmetauscher, und somit die Erhitzung, zu regeln wird vor den Wärmetauscher ein Ventil geschalten. Dieses Ventil regelt wie viel vom heissem Medium in den Wärmetauscher gelangt. Der Rest wird durch Rohre an dem Wärmetauscher vorbei geführt und trifft hinter diesem mit dem Ausgang des Wärmetauschers wieder zusammen.

Der Regler des Wärmetauschers steuert dieses Ventil und somit auch den Grad der Erhitzung des Mediums an der sekundär Seite. In dem

gegeben technischen System benützt der Wärmetauscher eine lineare Funktion um das Kondensat zu erhitzten. Somit steuert der Regler mit einem Prozentsatz der Anzahl des primären Mediums, wie stark das sekundäre Medium erhitzt wird.

4.2.6 Separator

Wie in 3.5 beschrieben dient ein Separator dazu, den Wasserdampf vom Kondensat zu trennen. Zudem dient er auch als Ausgangspunkt für den Dampf und als Eingangspunkt für den Nachschub an Kondensat.

Der Separator hat 4 Anschlüsse: zwei Eingänge und zwei Ausgänge. Der erste Ausgang verbindet den Saparator mit der Zirkulationspumpe, der erste Eingang wird mit dem Ventil verbunden, über den zweiten Ausgang wird der Dampf abgelassen und der zweite Eingang dient als Nachschubquelle an Kondensat.

Wie schon in 3.5.2 erklärt, dient der Separator auch als ein Speichermedium. Durch das Ventil kommt erhitztes Kondensat herein und vermischt sich mit dem vorhandenen Kondensat und auch mit dem nachkommenden Kondensat vom Nachschub-Eingang. Das vermischte Kondensat wird dann wieder durch die Zirkulationspumpe gepumpt. Der ankommende Anteil an Dampf wird an der Oberseite von dem Separator gesammelt und durch den zweiten Anschluss abgeführt. Somit sinkt auch der Füllstand des Separators und muss mit frischem Kondensat aufgefüllt werden. Als Kontrollgrösse wird der Füllstand gemessen, wenn der aktuelle Füllstand unter eine Minimum-Grenze fällt, wird mehr frisches Kondensat nach gepumpt.

Parameter

Folgende wichtige Parameter des Separators müssen gesetzt werden:

Fassungsvermögen gibt das maximale Fassungsvermögen in [m^3] an.
Anfangs-Druck in [bar].
Anfangs-Temperatur in [°C].
Anfangs-Füllstand in [%] des Fassungsvermögens.

4.2.7 Zeitfaktor

Alle Modelle sind mit einer Variable versehen um verschiedene Zeitabläufe auf einen gemeinsamen Nenner zu bringen. Jede Variable die auf eine Zeiteinheit bezogen ist, wird mit diesem Zeitfaktor multipliziert. Dies ist besonders wichtig um mit den verschiedenen Einheiten umzugehen. Zum Beispiel wird bei einem Wärmetauscher die Steigerung der Temperatur in $\Delta T = \frac{^\circ C}{s}$ berechnet. Genauso wie die Temperaturänderung im Separator auch in Sekunden berechnet wird.

Im Gegensatz dazu kann man den Durchfluss auch in $\frac{kg}{h}$ angeben, dazu muss entsprechend der Zeitfaktor angepasst werden.

4.3 Regler

Wie schon oben erwähnt , siehe Abschnitt 4.2.3 und 4.2.4, werden Regler benötigt um den Durchfluss und Druck zu regulieren. Ein Regler muss die Pumpe ansteuern um einen bestimmten Druck im Wärmetauscher erreichen zu können. Durch das Ventil kommt es zu einem Rückdruck in den Leitungen.

4.3.1 Ablauf

Als Regler werden Zweipunktregler verwendet. Durch Angabe eines Sollwertes und des aktuellen Wertes , liefert der Regler entweder 1 oder -1. Dieser Ausgang wird dann bei einer Pumpe benützt um stärker oder schwächer zu arbeiten. Bei einigen Regler wie zum Beispiel dem Wärmetauscherregler kann die Schrittgrösse als Parameter eingestellt werden und erlaubt so eine individuelle Anpassung an die jeweiligen Bedürfnisse. Durch diese Schrittgrößen kann man indirekt auch die Reaktionsgeschwindigkeit des betreffenden Elements einstellen. Dies erlaubt es reale Abläufe besser zu simulieren, da eine Pumpe nicht in einer Sekunde auf volle Leistung springen kann, sondern kontinuierlich anläuft.

4.3.2 Zirkulationspumpen-Regler

Der Pumpenregler ist ein einfacher Regler der nur einen Sollwert halten soll. Durch die Rückwirkung des Ventils kommt es zu einem neuen Druck der vom Sollwert abweicht, weshalb immer nachgeregelt werden muss. Bild 4.2 zeigt den Einschwingvorgang anhand des Beispiels des Pumpenreglers.

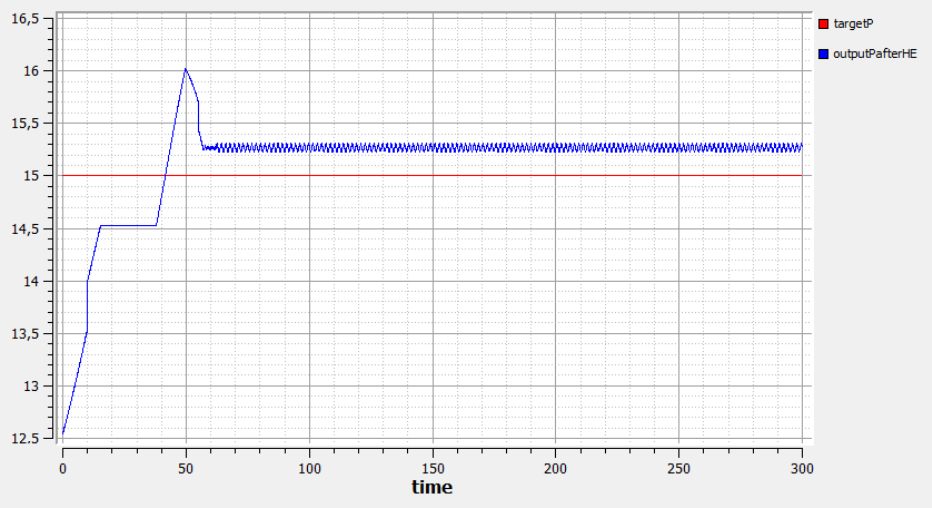


Abbildung 4.2: Ein Beispiel für einen Regler anhand des Einschwingen des Pumpenreglers.

4.3.3 Ventil-Regler

Beim Ventil ist die Vorgabe eine andere. Es soll zum einen ein Maximum an Dampf nach dem Ventil im Separator entstehen aber auch verhindert werden, dass Dampf im System entsteht. Deshalb achtet der Regler auf die Sättigungstemperatur jeweils vor und nach dem Ventil und versucht sich in einem Bereich einzustellen. Wie schon oben in 3.4 erläutert, kommt es durch das Schließen des Ventils zu einem Druckverlust nach dem Ventil. Dadurch sinkt auch die Sättigungstemperatur nach dem Ventil. Der Regler versucht den Druck so einzustellen, dass zum einen die aktuelle Temperatur über die Sättigungstemperatur nach dem Ventil gerät.

Zum anderen achtet er auch auf die Temperatur vor dem Ventil, damit diese unter der Sättigungstemperatur vor dem Ventil bleibt. Hierzu kann man einen Schwellwert einführen um den Abstand zur Sättigungstemperatur zu definieren. Dadurch werden Schwankungen der Regler im System abgefangen und es kommt zu keiner Verdampfung vor dem Ventil.

5 Ergebnisse

Nach einer Einführung in Modelica, der Erläuterung der technischen Elemente und der Beschreibung der Modelle werden in diesem Kapitel exemplarisch einige Ergebnisse gezeigt. Als Simulationssystem wird das wie in der Angabe in 3.2 beschriebene System verwendet. Ausgehend von verschiedenen Betriebspunkten wie zum Beispiel beim Anfahren des Systems, wird das Verhalten des Systems und der einzelnen Komponenten untersucht.

Per Definition ist der Fluss beim Eingang einer Komponente positiv. Dementsprechend muss der Fluss am Ausgang der Komponente negativ sein, da $inputPort.m + outletPort.m = 0$ ist. Bei einigen Beispielen wird der Mengenfluss als negativer Wert angezeigt da am Ausgang gemessen wird.

Die Kennlinien der Zirkulationspumpe wurden aus dem Datenblatt einer Grundfos CRN 45-5 Pumpe entnommen [Gru14, Page 61]. Die Kennlinien des Ventils entsprechen der eines RA 50 Ventils der Firma Metso [Met15]. Entsprechend der Kennlinien können diese Elemente einen begrenzten Arbeitsbereich abdecken.

5.1 Ein stabiles System

Als Einstieg wird das Verhalten des Systems ausgehend von einem stabilen Betriebspunkt betrachtet. Dabei soll das System von einem eingestellten Anfangszustand aus einen gewissen Zustand erreichen. In diesem Fall soll ein festgelegter Druck erreicht werden. Zur Simulation des Systems dienen reale Elemente als Vorlagen. Diese haben bestimmte Einsatzbereiche und Eigenschaften die das System beeinflussen. Abhängig

von diesen realen Einschränkungen wird ein stabiler Betriebspunkt berechnet von welchem das System startet.

5.1.1 Berechnung des Dampfes

Die wichtigste Berechnung ist die der Menge an generierten Dampf. Hierzu wird von einem Soll-Druck von 17.3 bar ausgegangen. Durch die Toleranz von 0.5 bar für den Pumpenregler kann ein maximaler Druck von 17.8 bar erreicht werden. Die Sättigungstemperatur bei diesem Druck liegt bei 206.5°C. Um ein Verdampfen in den Leitungen zu vermeiden wird eine Reserve von 5°C für die Regler festgelegt. Dadurch ergibt sich eine gewünschte Temperatur von 201.5°C. Hier noch einmal die wichtigsten Werte zur Berechnung der Menge an Dampf:

$$\begin{aligned} P_{target} &= 17.8bar \\ T &= 201.5°C \\ P_{Separator} &= 8bar \\ m &= 40000\frac{kg}{h} \end{aligned}$$

Für die Gleichung 4.2 werden die Werte der Enthalpie für des flüssigen Eingangsflusses und des flüssigen und des gasförmigen Anteils am Ausgang des Ventils benötigt. Die Enthalpie am Eingang lässt sich durch $h\_pt(P,T)$ des IAPWS 3.2 berechnen. Die Ausgangstemperatur ist die Verdampfungstemperatur bei 8 bar und kann durch $Tsat\_p(P_{out})$ berechnet werden.. Genauso lassen sich die Enthalpien mit $hV\_p(P_{out})$ und $hL\_p(P_{out})$ berechnen. Durch diese Werte kann man X und mit dem Eingangsfluss die Menge an Dampf, wie in der Gleichung 4.3 angegeben, berechnen. Ein Berechnungsbeispiel und die Ergebnisse sind in 5.1 ersichtlich.

$$h_{in} = h\_pt(17.8, 201.5) = 859.2287\frac{kJ}{kg}$$
$$h_{vapOut} = hV\_p(8) = 2768.3\frac{kJ}{kg}$$
$$h_{liquOut} = hL\_p(8) = 721\frac{kJ}{kg}$$
$$X = \frac{(h_{in} - h_{vapOut})}{(h_{liquOut} - h_{vapOut})} = \frac{(859.2287 - 2768.3)}{721 - 2768.3)} = 0.932491$$
$$mV = m_{in} * (1 - X) = 2700\frac{kg}{h} \quad (5.1)$$

5.1.2 Die Konfiguration

- Separator
  - Start Volumen = 50 %
  - Start Druck = 8.2 bar
  - Start Volumen = 50 m^3
  - Maximales Volumen = 100 m^3
- Ventil
  - C_v = 180
  - D = 0.05 m
- Pumpe
  - Leistung = 90 %
- Wärmetauscher
  - Leistung = 50 %
  - Q = 414 $\frac{kg*C}{h}$
- Simulationsparameter
  - Stoppzeit = 4000
  - Anzahl der Schritte = 4000
  - Optimierungsmethode = newuoa

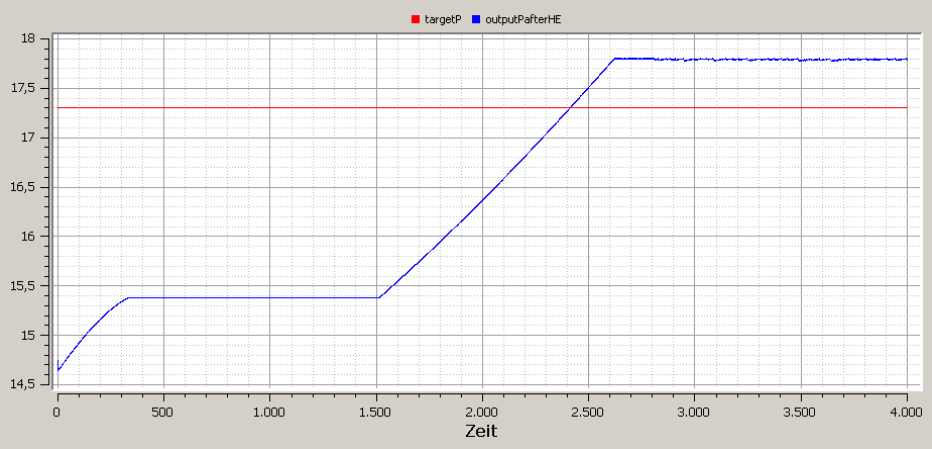


Abbildung 5.1: Der Druckverlauf bei einem stabilen Betriebspunkt. Der Solldruck von 17.3 bar wird ca. t = 2400 erreicht.

Die Pumpe startet mit einer Leistung von 90 % und steigert danach die Leistung um den Soll-Druck von 17.3 bar zu erreichen. Das Bild 5.1 zeigt den Druckverlauf während der Simulation. Nach t = 2400 wird der Soll-Druck überschritten und somit wird die Leistung wieder langsam gedrosselt wie in Bild 5.4 zu sehen ist . Bei dieser Leistung der Pumpe kann ein Druck von 17.7 bar erreicht werden. Dieser Druck befindet sich mit 0.5 bar Unterschied innerhalb der eingestellten Toleranz des Reglers.

Betrachtet wir nun den Bereich um t=1000. Man sieht, dass der Druck in diesem Bereich konstant bleibt obwohl der Soll-Druck noch nicht erreicht ist. Da der Regler schon bei 90% startet erreicht er schnell seine maximale Leistung und fördert die maximale Menge an Kondensat, wie im Bild 5.2 zu sehen ist. Das Ventil ist aber noch recht weit offen und somit ist der Gegendruck im System nicht sehr hoch. Das Bild 5.3 zeigt den Verlauf des Gegendruckes und den des neuen Druckes der Pumpe in Meter. Man erkennt, dass solange der Gegendruck unter dem Druck der Pumpe bleibt, kann diese die maximale Menge an Kondensat fördern. Sobald diese Grenze überschritten wird, bei ca. t=1500, verschiebt sich die Anlagenkennlinie und der Druck steigt und die Menge an gefördertem Kondensat sinkt.

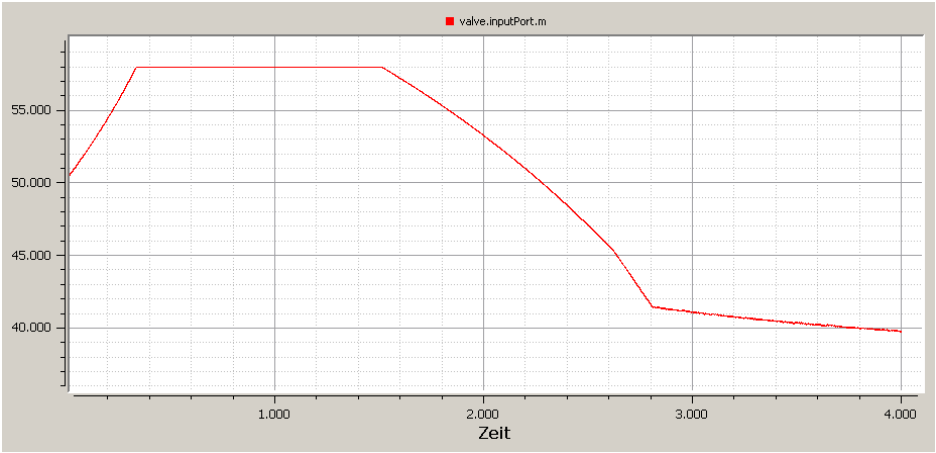


Abbildung 5.2: Die Menge an gefördertem Kondensat.

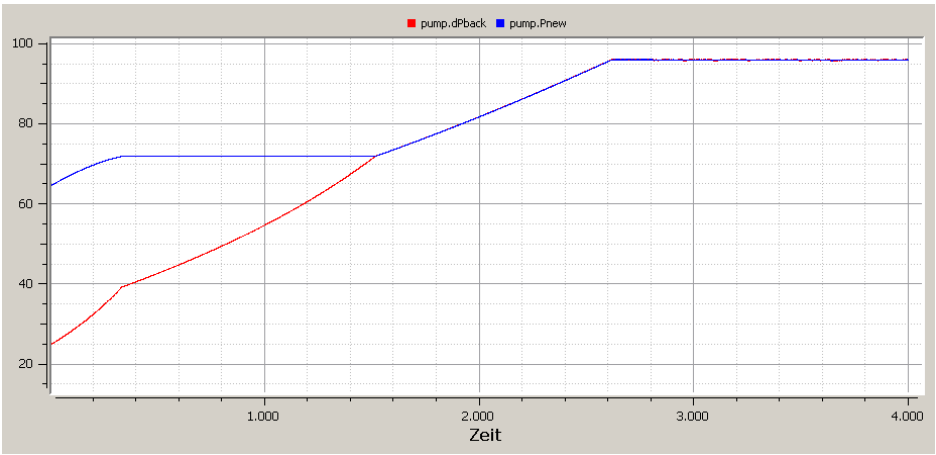


Abbildung 5.3: Der Druckverlauf des Gegendruckes in rot und der Pumpe in Meter.

Das Bild 5.4 liefert viele Informationen über das Verhalten der Regler. Die Linien zeigen den Öffnungsgrad des Ventils, die prozentuale Leistung des Wärmetauschers und die Leistung der Pumpe. Besonders in Verbindung mit dem Bild 5.5 kann man viele Rückschlüsse ziehen und Verbesserungen an den Reglern vornehmen. Man sieht dort die ak-

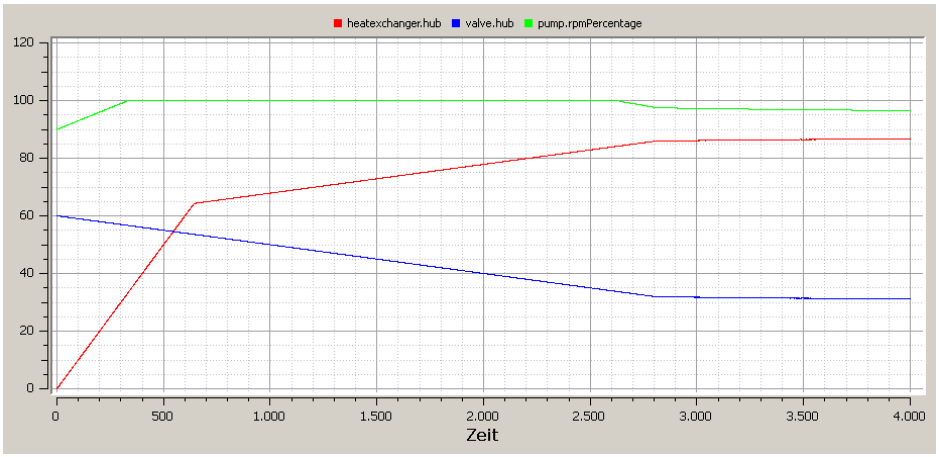


Abbildung 5.4: Der Öffnungsgrad und die Leistung der Pumpe, des Wärmetauscher und Ventils.

tuelle Temperatur, gemessen nach dem Wärmetauscher, in grün. Die rote Linie ist die Verdampfungstemperatur in den Leitungen vor dem Ventil. Die blaue Linie die selbe Temperatur nach dem Ventil, also nach dessen Druckverlust und somit einer geringeren Verdampfungstemperatur.

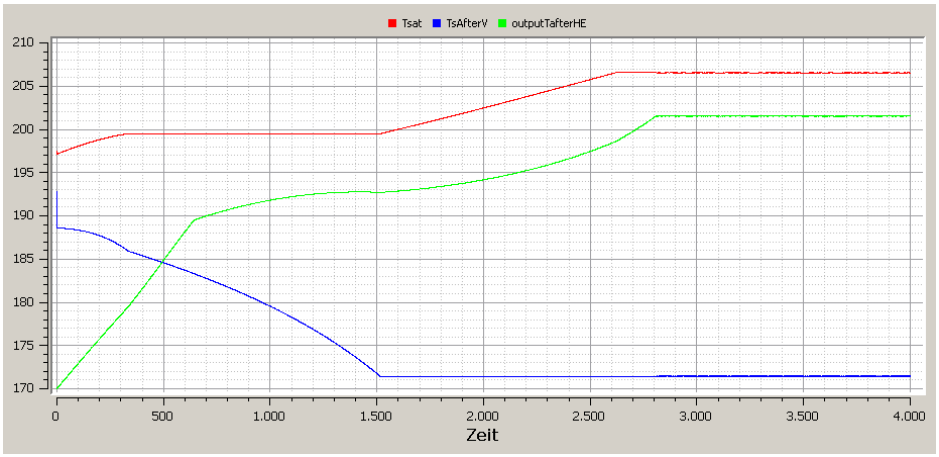


Abbildung 5.5: Die Verdampfungstemperatur vor und nach dem Ventil mit dessen Eingangstemperatur.

Dass die Pumpe schnell ihre maximale Leistung erreicht wissen wir schon. Das Ventil ist mit 60% noch recht weit offen und beginnt sich langsam zu schließen. Die größte Auswirkung hat am Anfang der Wärmetauscher der schnell seine Leistung steigert und die Temperatur erhöht. Bei t $\approx$ 650 erreicht die Temperatur die Schwelle von dem doppelten eingestellten Sicherheitsabstand von 5° C und der Regler des Wärmetauschers verwendet eine kleinere Schrittweite. Dadurch steigt die Temperatur ab diesem Zeitpunkt langsamer.

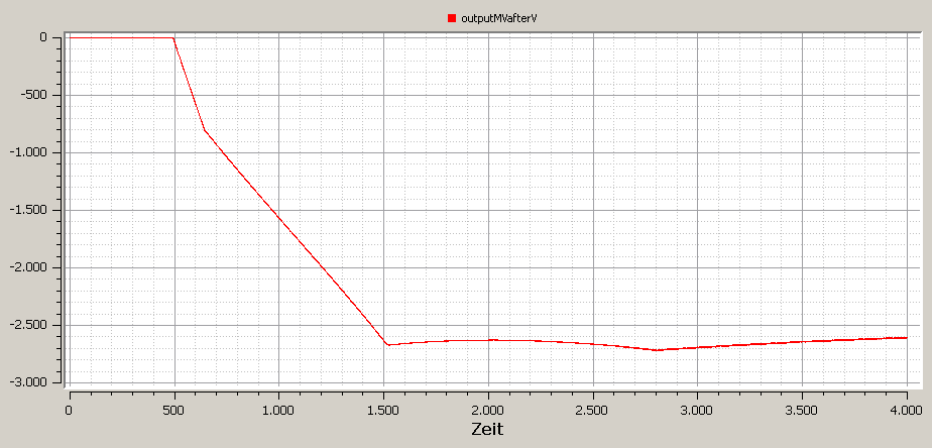


Abbildung 5.6: Die Menge an resultierendem Dampf.

Im Vergleich mit dem Bild 5.6 sieht man, dass sobald die Temperatur über die Verdampfungstemperatur nach dem Ventil steigt, durch den Druckverlust des Ventils Dampf erzeugt wird. Der Anstieg der Temperatur erlaubt es mehr Dampf zu erzeugen. Wenn die Temperatur ungefähr 5 °C unter der Verdampfungstemperatur liegt schwingen sich alle Regler ein und die Temperatur bleibt fast konstant.

Das Verdampfen verbraucht Energie und dadurch sinkt die Temperatur des übriggebliebenen Kondensates. Diese sieht man in Bild 5.7 als blaue Linie. Zuerst steigt diese Linie, wie oben beschrieben, sinkt dann aber entlang der Sättigungstemperatur des Kondensates nach dem Ventil. Die rote Linie zeigt die Temperatur im Separator, die durch den Rückfluss des eben genannten, heißen Kondensates und den Zufluss von neuem, kalten Kondensat beeinflusst wird. Nach einem Erhitzten sinkt die Tem-

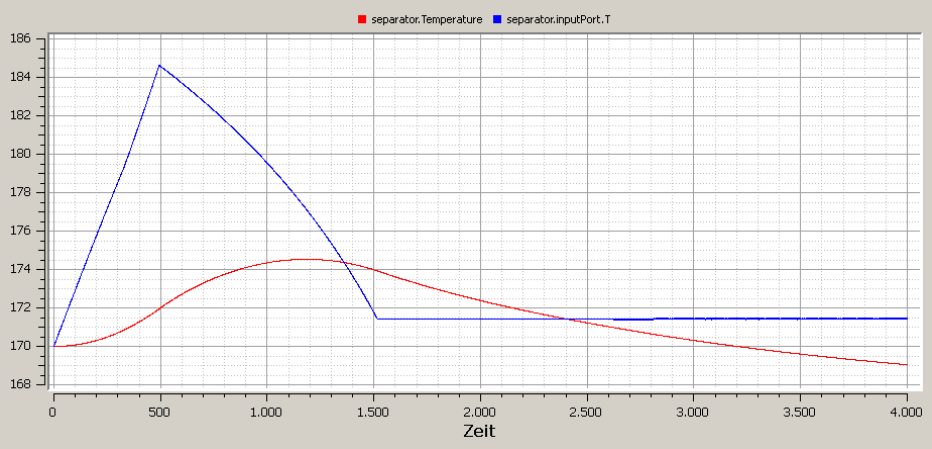


Abbildung 5.7: Der Temperaturverlauf im Separator und des Ausgangs des Ventils.

peratur wieder, da der Zufluss von kaltem Kondensat die Temperatur senkt.

Die Ergebnisse zeigen, dass die Regler gut und das System richtig arbeitet. Nach dem Einschwingvorgang wird im Mittel 2700 $\frac{kg}{h}$ Dampf erzeugt. Dieses Ergebnis stimmt mit den zuvor berechneten Werten in 5.1 überein.

Die realen Daten basieren auf einem Auslegungsprogramm und nicht genauen Messwerten. In diesem Auslegungsprogramm ist die Basis zur Berechnung der Menge an Dampf eine Kombination aus der Enthalpien nach dem Ventil und der Menge an Energie die durch den Wärmetauscher maximal gewonnen werden kann. Die Berechnung der Enthalpien ist schon von oben bekannt und ist abhängig vom eingestellten Druck im Separator. Die Menge der gewonnen Energie durch den Wärmetauscher wird durch die Eingangs- und Ausgangstemperatur auf der Gasseite berechnet. Die dadurch berechnete Menge an Dampf ist für diesen stabilen Betriebspunkt konstant und entspricht mit 2641$\frac{kg}{h}$ den hier simulierten Werten. Schwankungen bei den Ergebniswerten sind auf das Schwingverhalten der Regler zurückzuführen.

Dieses stabile System kann dazu verwendet werden die Regler einzustellen. Je nach Einstellung der Regler kann die Geschwindigkeit und

Stabilität des Einschwingvorganges stark beeinflusst werden.

5.2 Anfahren des Systems

Nachdem das stabile System besprochen wurde und auch das Verhalten zu diesem Betriebspunkt analysiert werden konnte, ist es an der Zeit das System in Grenzsituationen zu analysieren. Das bisher erläuterte System behandelt den normalen Fall in einem gut eingestellten System mit stabilen Betriebspunkt. Ausgehend von diesem stabilen System kann man das System unter anderen Blickwinkeln betrachten und dessen Verhalten bei verschiedenen Betriebspunkten beobachten. Ein sehr wichtiger Fall in der Praxis ist das Verhalten im Anfangszustand. Hierzu ist es nicht nur nötig die Anfangszustände zu ändern, sondern auch die Regler anzupassen.

Anders als oben in 5.1 werden bei der Simulation des Anfahrens des Systems einige Anfangsparameter geändert. Dazu gehört die Anfangsbedienung, dass das System leer, also ohne Kondensat, ist. Daher ist auch der Separator am Anfang leer, das Ventil voll aufgedreht und die Pumpe ausgeschaltet. Der Separator wird mit der Zeit, durch frisches Kondensat, aufgefüllt und ab einem einstellbaren minimalen Füllstand wird das System in Betrieb genommen. Zur Vergleichszwecken werden dabei die gleichen Komponenten wie bisher verwendet.

- Separator
  - Start Volumen = 19.1 %
  - Start Druck = 8.2 bar
  - Start Volumen = 7.2 m^3
  - Maximales Volumen = 37.6 m^3
- Ventil
  - C_v = 180
  - D = 0.05 m
  - Leistung = 100 %
- Pumpe
  - Leistung = 0 %

- Wärmetauscher
  - Leistung = 0 %
  - Q = 414 $\frac{kg*C}{h}$
- Simulationsparameter
  - Stoppzeit = 8000
  - Anzahl der Schritte = 8000
  - Optimierungsmethode = newuoa

Das bisherige Modell entspricht nicht ganz diesen Anforderungen und so wurden einige Änderungen hinzugefügt. Ein Problem ist der Anfangsfluss der Pumpe. Die Pumpe braucht einen fixen Druck auf der Eingangseite um arbeiten zu können, zu sehen an Pumpenkennlinie im Bild 3.4. Dieser Druck wird in einem stabilen System als Anfangsparameter eingegeben. Im stabilen System hat die Pumpe die Menge an Kondensat bestimmt, die aus dem Separator gezogen wird. Eine ausgeschaltete Pumpe kann keinen Fluss ziehen und so muss der Ausfluss aus dem Separator neu berechnet werden.

Die Menge an Kondensat im Separator bestimmt die Menge an Kondensat das durch die Ausflussöffnung in Richtung Pumpe fließt. Wie schon vorher in Abschnitt 3.5 beschrieben, lässt sich der neue Fluss durch die Füllhöhe und der Fläche des Ausflusses berechnen. Die Gleichung 3.14 lässt sich durch die Annahme vereinfachen, dass der Druck im Separator gleich dem Ausflussdruck ist. Dies ist legitim, da das System von diesem Ausfluss bis zum Einfluss in den Separator geschlossen ist und die Pumpe keine Druckdifferenz erzeugt. Somit ergibt sich die vereinfachten Formel 5.2 und dadurch der neue Fluss wie in der Formel 3.15 angegeben.

$$c_a = \sqrt{2 * g * h} \tag{5.2}$$

Das Modell der Pumpe wurde so verändert, dass sie entweder einen eigenen Fluss zieht oder geöffnet ist und somit den Fluss des Separators unverändert durchfließen lässt. Nun werden einige Ergebnisse des Anfahrens gezeigt, die für diesen Betriebspunkt charakteristisch sind.

Bisher war die Menge an frischem Kondensat an die Menge des generierten Dampfes gekoppelt und dadurch blieb der Füllstand konstant.

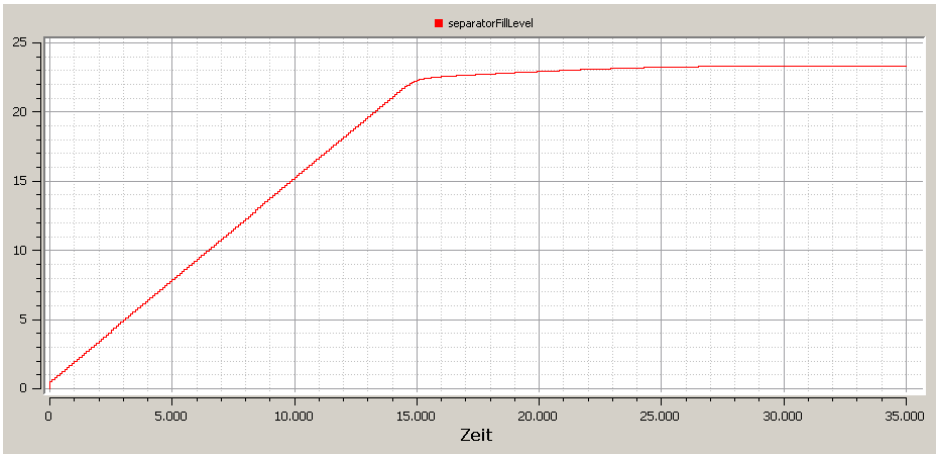


Abbildung 5.8: Der Füllstand des Separators steigt beim Anfahren und erreicht bei t=13000 die 20% Grenze.

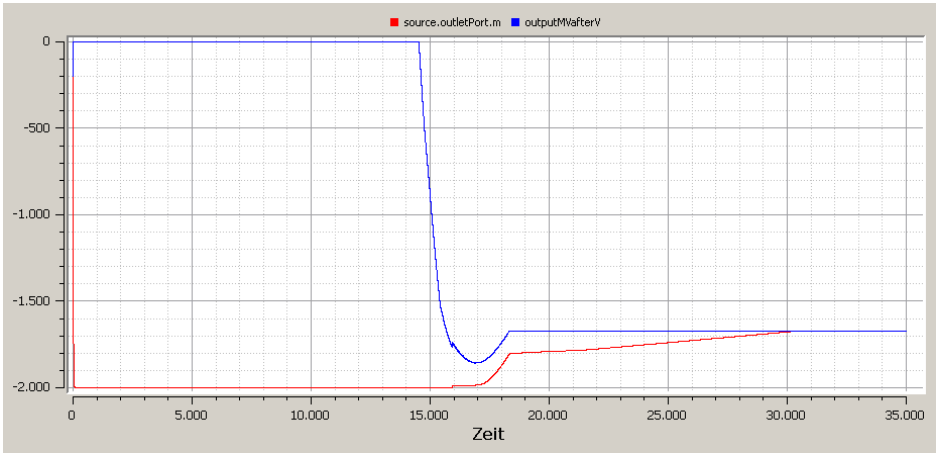
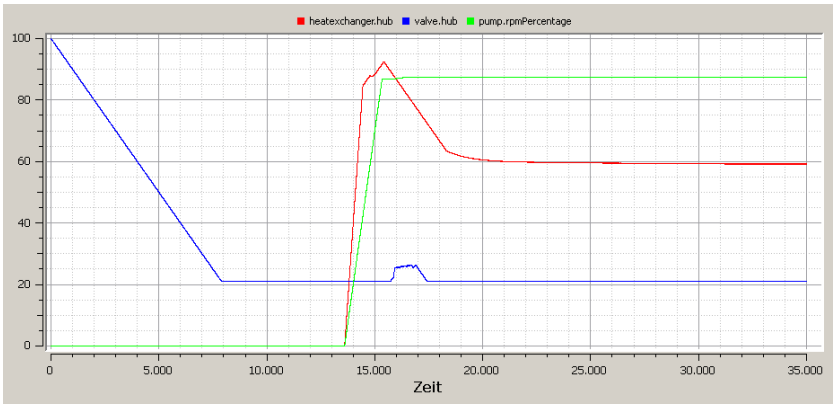
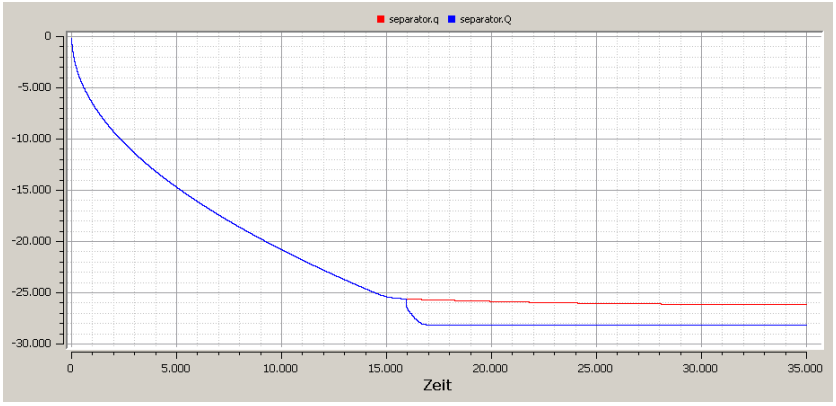


Abbildung 5.9: Die Menge an generiertem Dampf und des frischen Kondensates beim Anfahren.

Nun muss eine Regelung eingebaut werden, damit auch Kondensat zuströmt wenn kein Dampf erzeugt wird. Somit wird aus einer einfachen Gleichung: $source.outletPort.m = -outputMV$ ein Regler der die Menge an frischem Kondensat steuern muss.



(a) Der Öffnungsgrad und die Leistung der Pumpe, des Wärmetauscher und Ventils



(b) Die Ausflussmenge im Separator.

Abbildung 5.10: Interressante Werte beim Anfahren.

Das Bild 5.8 zeigt den Füllstand des Separators. Nach dem Befüllen wird bei t $\approx$ 13000 die 20% Marke überschritten und die Menge an frischem Kondensat wird gedrosselt, wie in Bild 5.9 an der roten Linie zu sehen ist. Danach gleicht sich diese an die Menge des generierten Dampfes an bis bei t$\approx$ 30.000 die Menge konstant bleibt.

Im Bild 5.11 werden die wichtigsten Temperaturverläufe gezeigt. Die Temperatur nach dem Wärmetauscher(die güne Linie), da dieser ausgeschaltet ist, gleich der Temperatur im Separator bis das System startet. Diese anfängliche Temperatur fällt auf die Temperatur des Zuflusses an

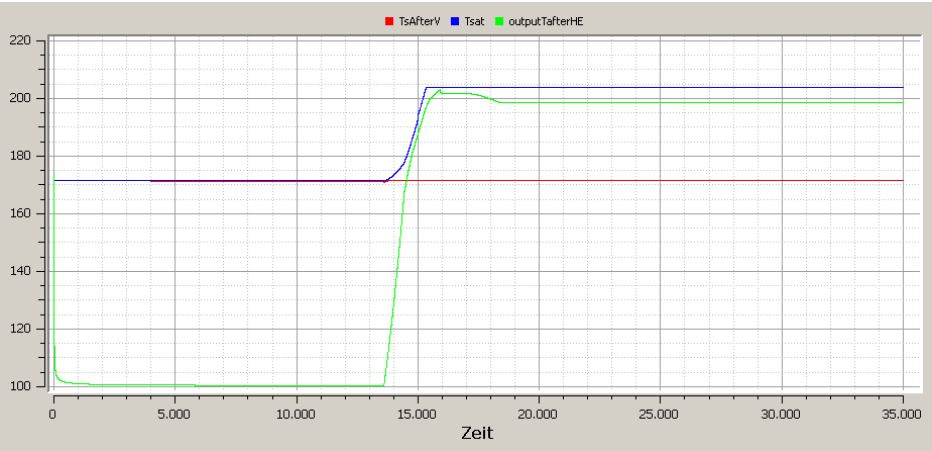


Abbildung 5.11: Die Temperaturverläufe beim Anfahren.

frischem Kondensat von $T = 100°C$. Nach dem Einschalten steigt der Druck im System und dadurch auch die Verdampfungstemperatur(blaue Linie) und auch der Wärmetauscher erhitzt das Kondensat.

Im Bild 5.10(b) zeigt einen Vergleich zwischen der Menge an Kondensat die den Separator verlässt, die blaue Linie, und die Menge die nach der Formel 5.2 berechnet wird, die rote Linie. Zuerst sind beide gleich, da der Fluss nur vom Füllstand des Separators abhängig ist. Dieser steigt immer mehr an, wodurch auch der Ausfluss größer wird. Bei $t \approx 15000$ hat die Pumpe genug Leistung erreicht um eine größere Menge an Kondensat zu ziehen als durch den Füllstand im Separator ermöglicht wird. Ab diesem Zeitpunkt wird der Ausfluss durch die Pumpe bestimmt.

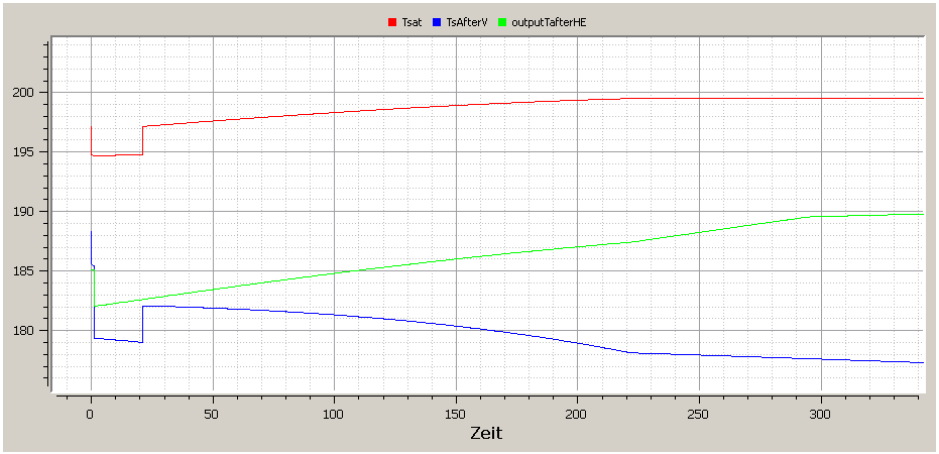
Die Änderung des Systems erwies sich als aufwendiger als erwartet. Dabei waren nicht die Änderungen an den Modellen das Problem, vielmehr wurden die Auswirkungen der Regler- und Simulationsparameter sichtbar. Die Modelle blieben weitgehend unverändert, bis auf die genannten Beispiele. Die Einstellungen der Regler beeinflussten stark das System, von der Menge an generiertem Dampf bis zum kompletten Absturz der Simulation. Genauso beeinflussen die Simulationsparameter das System sehr stark.

OpenModelica erlaubt es sowohl den Zeitabschnitt der simuliert werden soll, als auch die Anzahl der Simulationsschritte zu setzen. Durch die

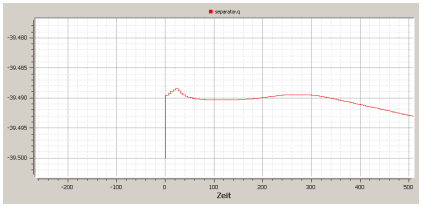
Verwendung von Differenzialgleichungen zur Beschreibung des Systems ist die richtige Wahl der Anzahl der Schritte wichtig. Zum Beispiel wird die Änderung der Leistung des Wärmetauschers auch in $\frac{\delta hub}{\delta t}$ angegeben. Werden zu wenige Simulationsschritte verwendet, wodurch die Schrittlänge steigt, kann das Kondensat zu stark erhitzt werden, bevor der Regler reagieren kann. Dieses bekannte Verhalten kann durch das Abtasttheorem behoben werden.

5.3 Erfahrungen und Erkenntnisse

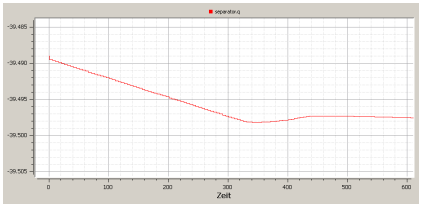
Wie jedem Modellierer bewusst ist, sind die Anfangswerte für die Simulation besonders wichtig. Die Auswirkungen dieser können das System instabil machen oder führen zu Ausreißern in den Ergebnissen. Ein Beispiel hierfür möchte ich gerne zeigen. Im Zuge dieser Arbeit sind oft Ergebnisse aufgetaucht, die starke Anomalien aufwiesen. Ein Beispiel ist im Bild 5.12(a) zu sehen. Die Sprünge am Anfang der Temperaturverläufe sind augenscheinlich falsch doch die Ursache kann an vielen Stellen liegen: Es kann der Wärmetauscher falsch eingestellt sein und zu wenig oder zu viel heizen, der Fluss kann selber einen solchen Sprung haben oder die Temperatur im Separator wurde nicht richtig berechnet. Nachdem die Ursache gefunden wurde, es waren falsche Anfangswerte für den vom Separator erzeugten Fluss wie in Bild 5.12(b) zu sehen ist, konnte der Sprung in der Temperatur beseitigt werden 5.12(c).



(a) Die Sprünge am Beginn sind durch schlechte Anfangswerte entstanden.



(b) Der Sprung am Anfang kommt von einem schlechten Anfangswert.



(c) Besserer Anfangswert.

Abbildung 5.12: Störungen aufgrund schlechter Anfangswerte.

5.4 Weitere Arbeit

Nachdem die Ergebnisse dieser Arbeit im letzten Abschnitt präsentiert wurden, werden hier einige Verbesserungsvorschläge erwähnt. Ziel dieser Arbeit ist es die praktische Anwendbarkeit von OpenModelica zu untersuchen. Für die praktische Anwendbarkeit wurden wie üblich einige Vereinfachungen vorgenommen.

Wie schon in 3.6 erwähnt, gibt es viele Arten von Wärmetauscher in unterschiedlichen Bauformen. Für die praktische Anwendung kann das Modell dahingehend vereinfacht werden, dass nur die maximale Wärmeleistung berücksichtigt wird. Eine Verbesserung des Systems könnte darin liegen, ein detailliertes Modell eines Wärmetauschers zu entwickeln. Dieses Modell sollte mehrere Bauformen abdecken und dabei ebenfalls leicht zu konfigurieren sein. Die Schnittstellen nach außen wie die Anschlüsse und die Verbindung zum Regler bleiben dabei gleich, nur die interne Berechnung der Temperaturdifferenz $\delta T^{\circ}C$ wird vom Modell geändert.

Für ein komplexeres System kann noch ein Modell der Verbindungsleitungen zwischen den Komponenten erstellt werden. So ein Modell einer Leitung müsste die Druckveränderung durch die Leitungen berücksichtigen. Dieser Druckverlust ζ lässt sich durch die mittlere Strömungsgeschwindigkeit, der Rohrreibungszahl λ und den geometrischen Größen (Länge und Durchmesser) der Leitung berechnen [GC05]. Aus der Rohreibungszahl und der mittleren Strömungsgeschwindigkeit lassen sich die Druckverlustbeiwerte der Leitungsstücke zu den Beiwerten der Armaturen hinzufügen und ergeben den neuen Druckverlust im gesamten System. Die Information über den Druckverlust muss dann noch an die Pumpe weitergegeben werden um die neue Anlagenkennlinie zu berechnen. Ein weiteres Attribut einer Leitung ist auch noch die Angabe der nötigen Menge an Kondensat um das Teilstück zu befüllen.

6 Zusammenfassung

Modelica bietet viele Vorteile zur Simulation eines technischen Systems. Die Verwendung von Formeln zur Modellbeschreibung ermöglicht die vielfältige Verwendung. Die Definition von bestimmten Parametern erlaubt es verschiedene Betriebspunkte einfach zu konfigurieren. Ohne die Modelle zu verändern kann so das Verhalten des modellierten Systems dynamisch unter verschiedenen Bedingungen analysiert werden. Es muss nicht das Verhalten zu verschiedenen Betriebspunkten separat modelliert werden. Dadurch können auch Nutzer mit eingeschränkten Programmierkenntnissen das Modell verwenden und analysieren.

Die Validierung der Ergebnisse mit den gegebenen Werten belegt die Funktionalität der einzelnen Komponenten und des gesamten Systems. Die Menge an generierten Dampf stimmt mit den Vergleichswerten überein.

Als Hindernis gestalten sich sehr verschachtelte Systeme wie es hier der Fall war. Durch die vielen Abhängigkeiten kann man Fehler schwer ausfindig machen und ausbessern. Als Beispiel hierfür dient die Modellierung des Anfahrens des Systems. Durch das Ändern der Rahmenbedingungen wird auch ein Anpassen der Regler nötig. Das System und die Modelle bleiben dabei unverändert. Dadurch kann man die Auswirkungen der Regler analysieren und das Verhalten anpassen.

Zusammenfassend ist Modelica eine gute Entwicklungsumgebung zum Erstellen von Modellen. Modelle lassen sich schnell implementieren, nur bei sehr verschachtelten Systemen wird die Fehlersuche komplexer.

Bibliography

[AT10] Syed Adeel Asghar and Sonia Tariq. Design and implementation of a user friendly openmodelica graphical connection editor, 2010.

[BB10] L. Böswirth and S. Bschorer. *Technische Strömungslehre: Lehr- und Übungsbuch ; mit 43 Tabellen*. Vieweg + Teubner, 2010.

[DS07] Surek Dominik and Stempin Silke. *Angewandte Strömungsmechanik - für Praxis und Studium*. Springer, Berlin, Heidelberg, 2007. aufl. edition, 2007.

[EMO99] H. Elmqvist, S.E. Mattsson, and M. Otter. Modelica-a language for physical system modeling, visualization and interaction. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 630–639, 1999.

[FR03] Joachim Rückert und Axel Schloßer Felix Richert. Vergleich von modelica und matlab anhand der modellbildung eines dieselmotors. *AT - Automatisierungstechnik*, 2003.

[Fri04] Peter Fritzson. *Principles of Object-Oreinted Modelling and Simulation with Modelica 2.1*. 2004.

[GC05] V. Gesellschaft and V.D.I.G.V.U. Chemieingenieurwesen. *VDI-Wärmeatlas*. Springer, 2005.

[Gru14] Grundfos. Grundfos datenheft: Cr, cri, crn, cre, crie, crne - vertikale mehrstufige kreiselpumpen. http://net.grundfos.com/Appl/WebCAPS/Grundfosliterature-1257.pdf, April 2014.

[Her04] Ralph Herbrich. *Stellventile* -. Oldenbourg Industrieverlag, München, 2004.

[IAP07] IAPWS. Revised release on the iapws industrial formulation 1997 for the thermodynamic properties of water and steam. The International Association for the Properties of Water and Steam, August 2007.

[Küm07] W. Kümmel. *Technische Strömungsmechanik: Theorie und Praxis*. Vieweg+Teubner Verlag, 2007.

[Luc05] Klaus Lucas. *Thermodynamik: Die Grundgesetze der Energie- und Stoffumwandlungen*. Springer, 5 edition, Sept 2005.

[Mat97] Sven Erik Mattsson. On modeling of heat exchangers in modelica. In *Proceedings of the 9th European Simulation Symposium, ESS'97*, October 1997.

[Met15] Metso. Neles ra series v-port segment valve. http://valveproducts.metso.com/neles/TechnicalBulletins/en/3R21EN.pdf, February 2015.

[Mod09] *Stream Connectors - An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena*, September 2009.

[mod14] *www.modelica.org*. 2014.

[NG05] C. Nytsch-Geusen. *MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. Proceedings of the 4th International Modelica Conference TU Hamburg-Harburg*. März 2005.

[SCHS00] Peter Schwarz, Christoph Clauß, Joachim Haase, and André Schneider. Modelica - eine objektorientierte beschreibungssprache für mechatronische systeme. *SIM2000 - Simulation im Maschinenbau*, page 83, 2000. Fraunhofer-Institut für Integrierte Schaltungen, Außenstelle für Automatisierung des Schaltungs- und Systementwurfs Zeunerstraße 38, D-01069 Dresden.

[SHJ97] Mattsson S.E., Elmqvist H., and Broenink J.F. Modelica: An international effort to design the next generation modelling language. *Journal A, Benelux Quarterly Journal on Automatic Control*, 38:3:16–19, September 1997.

[Sho15] Tom Short. *OpenModelica models in JavaScript*. https://github.com/tshort/openmodelica-javascript, feb 2015.

[Thi15] Michael Thiller. *Modelica by Example*. http://book.xogeny.com/, feb 2015. HTML Version only.

[vB04] Peter von Böckh. *Fluidmechanik*. Springer-Lehrbuch. Springer, 2004.

A Code

Code A.1: Das System-Modell.

```
model SystemStartUp
    import MA;
    SpecificModels.WEPA_HeatExchanger heatexchanger(Q = 414, timeFactor = parameterTimeFactor);
    SpecificModels.ValveCV180 valve(minPout = parametersourceP, timeFactor = parameterTimeFactor);
    SpecificModels.WEPA_ZP pump(timeFactor = parameterTimeFactor, rho = 1000);
    Models.Outlet outlet;
    Models.endlessSource source(SetTemperature = parametersourceT, P = parametersourceP,
            mV = parametersourceMV, timeFactor = parameterTimeFactor);
    MA.SpecificModels.WEPA_Seperator separator(startq = parameterSeparatorStartq, height = 3,
             area = 3.14 * SepD * SepD / 4, timeFactor = parameterTimeFactor, startPressure = 8.2,
             startTemperature = 170, startVolume = parameterSeparatorVolume);
    parameter Real parameterTimeFactor = 1 "35000 steps,newuoa";
    parameter Real SepD = 4;
    parameter Real targetP = 17.3;
    parameter Real parametersourceMV = 0,parametersourceT = 100,parametersourceP = 8.2;
    parameter Real parameterSeparatorVolume = 0.001;
    parameter Real parameterTempDiffMin = 5;
    parameter Real parameterSeparatorStartq = -200;
    Real controlledSourceM;
    Real sourceMMax;
    discrete Real separatorFillLevel;
    Real controlledHubPump(start = 0, min = 0.0000001, final max = 100.0);
    Real controlledHubValve(start = 100, min = 0.0000001, final max = 100.0);
    Real controlledHubHE(start = 0, min = 0.0000001, final max = 100.0);
    Real derHubPump(start = 1),derHubValve(start = 1),derHubHeatexchanger(start = 0.1);
    parameter Real ValveThresh = 2;
    parameter Real parameterValveBigStep = 0.05;
    parameter Real parameterValveSmallStep = 0.01;
    parameter Real parameterBigStep = 0.1;
    parameter Real parameterSmallStep = 0.01;
    parameter Real parameterSourceMDefault = 2000;
    parameter Real parameterPumpStep = 0.05;
    Boolean startPump(start = false);
    Real Tsat,TsAfterV;
    Real derValveHub;
    discrete Real vap(start = 0.0);
    Real derSourceMHelper(start = 0.0),derSourceM(start = 0.0);
    Real outputPafterHE,outputTafterHE,outputMafterHE,outputTafterV,outputPafterV,outputMVafterV;
    Real outputPreTHE,outputTsat,outputTsA,ControllerValveState;
  initial equation
    pump.run = startPump;
    pump.rpmPercentage = 0;
    valve.hub = 100;
    valve.run = false;
    separator.selfFlow = not startPump;
    controlledSourceM = 1;
  equation
    connect(source.outletPort,separator.inputPort2);
    connect(separator.outletPort,pump.inputPort);
    connect(pump.outletPort,heatexchanger.inputPort);
    connect(heatexchanger.outletPort,valve.inputPort);
    connect(valve.outletPort,separator.inputPort);
```

```
53      connect(separator.outletPort2,outlet.inputPort);
54      connect(valve.pressurePort,pump.pressurePort);
55      when sample(1, 100) then
56            separatorFillLevel = 100 * separator.Volume / separator.parameterMaxVol;
57
58      end when;
59      when sample(10, 10) then
60            vap = (pre(vap) + separator.outletPort2.mV) / 2;
61        derSourceMHelper = (vap - pre(vap)) / 10;
62
63      end when;
64      if separatorFillLevel > 20 then
65        pump.run = true;
66        separator.selfFlow = false;
67        startPump = true;
68        sourceMMax = -separator.outletPort2.mV;
69        derSourceM = -derSourceMHelper;
70        derHubPump = Controller.ControllerPump(pre(outputPafterHE), targetP,
71              pre(controlledHubPump), 0.5, parameterPumpStep);
72        derHubHeatexchanger = controlledHubHE;
73        derValveHub = der(controlledHubValve);
74        valve.run = true;
75      else
76        valve.run = false;
77        startPump = false;
78        pump.run = false;
79        separator.selfFlow = true;
80        derSourceM = (parameterSourceMDefault - controlledSourceM) / 10;
81        sourceMMax = parameterSourceMDefault;
82        derHubPump = -1;
83        derHubHeatexchanger = 0;
84        derValveHub = 0;
85      end if;
86      valve.hub = controlledHubValve;
87      der(controlledHubValve) = Functions.Utility.Range(controlledHubValve, 21, 100, derHubValve);
88      der(controlledSourceM) = Functions.Utility.Range(controlledSourceM, 0, sourceMMax, derSourceM);
89      separator.mDrawn = pump.mDrawn;
90      der(controlledHubPump) = Functions.Utility.Range(controlledHubPump, 0, 100, derHubPump);
91      valve.pressurePort.flow_ = 0.0;
92      pump.rpmPercentage = controlledHubPump;
93      Tsat = Functions.Tsat_p(outputPafterHE);
94      TsAfterV = Functions.Tsat_p(outputPafterV);
95      outputPafterHE = heatexchanger.outletPort.P;
96      outputTafterHE = heatexchanger.outletPort.T;
97      outputMafterHE = heatexchanger.outletPort.m;
98      outputTafterV = valve.outletPort.T;
99      outputPafterV = valve.outletPort.P;
100     outputMVafterV = valve.outletPort.mV;
101     source.SetMassflow = controlledSourceM;
102     outputTsat = Tsat;
103     outputTsA = TsAfterV;
104     outputPreTHE = pre(outputTafterHE);
105     (derHubValve,ControllerValveState) = Controller.ControllerValve(pre(outputTafterHE), pre(outputPafterHE),
106                   ValveThresh, pre(valve.pressurePort.dP), parameterValveBigStep, parameterValveSmallStep);
107     der(heatexchanger.hub) = derHubHeatexchanger;
108     controlledHubHE = Controller.ControllerHeatexchanger(pre(heatexchanger.outletPort.T), pre(Tsat),
109               parameterBigStep, parameterSmallStep, 0, 100, -parameterTempDiffMin, pre(heatexchanger.hub));
110     annotation(experiment(StartTime = 0, StopTime = 35000, Tolerance = 0.01));
111 end SystemStartUp;
```

Code A.2: Die Basis-Modelle.

```
1  package Models
2      connector StreamConnector
3        flow Real m "unit = [kg/h]";
4        Real P "unit = [bar]";
5        Real T "unit = [C]";
6        flow Real mV "unit = [kg/h]";
7      end StreamConnector;
8      model endlessSource
9        Real SetMassflow;
10       Real SetTemperature;
11       parameter Real P,mV;
12       parameter Real timeFactor = 1;
```

```
13      Models.StreamConnector outletPort;
14    equation
15      outletPort.m = -SetMassflow / timeFactor;
16      outletPort.mV = -mV / timeFactor;
17      outletPort.P = P;
18      outletPort.T = SetTemperature;
19    end endlessSource;
20    model Outlet
21      Real m,P,T,mV;
22      Models.StreamConnector inputPort;
23    equation
24      m = inputPort.m;
25      P = inputPort.P;
26      mV = inputPort.mV;
27      T = inputPort.T;
28    end Outlet;
29    partial model TwoPort
30      Models.StreamConnector inputPort;
31      Models.StreamConnector outletPort;
32    protected
33      Real dT(start = 0);
34      Real dP(start = 0);
35    equation
36      outletPort.T = inputPort.T + dT;
37      outletPort.P = inputPort.P + dP;
38      outletPort.mV + inputPort.mV = 0;
39      outletPort.m + inputPort.m = 0;
40      annotation(Icon(), Diagram());
41    end TwoPort;
42    partial model SpecialTwoPort
43      Models.StreamConnector outletPort;
44      Models.StreamConnector inputPort;
45    protected
46      Real dT(start = 0);
47      Real dP(start = 0);
48      Real dmV(start = 0);
49      Real dm(start = 0);
50    equation
51      outletPort.T = inputPort.T + dT;
52      outletPort.P = inputPort.P + dP;
53      outletPort.mV + inputPort.mV - dmV = 0;
54      outletPort.m + inputPort.m - dm = 0;
55    end SpecialTwoPort;
56    partial model HeatExchanger
57      extends Models.TwoPort;
58      parameter Real Q(unit = "kg*C/h = kW");
59      Real hub(min = 0, max = 100, unit = "[%]");
60      parameter Real timeFactor(unit = "1/h");
61    equation
62      dT = Q * hub / 100 * 3600 / inputPort.m / timeFactor;
63      dP = 0;
64    end HeatExchanger;
65    partial model Pump
66      extends Models.TwoPort;
67      import MA;
68      Real Vnew(start = 10.0, unit = "m3/h");
69      Real Pnew(final min = 0.0000001, unit = "m");
70      Real X2;
71      Real Vold;
72      Real dPback(unit = "m");
73      Real rpmPercentage(unit = "%", final min = 0.0000001, final max = 100);
74      Real Vmax;
75      parameter Real rho(unit = "kg/m3", start = 1000);
76      MA.Models.BackPressureConnector pressurePort;
77    equation
78      dPback = -10 * pressurePort.dP;
79    end Pump;
80    partial model Valve
81      extends SpecialTwoPort;
82      parameter Real timeFactor;
83      parameter Real Cv;
84      parameter Real rho(unit = "kg/m3");
85      parameter Real D(unit = "[m]");
86      Real hub(unit = "[%]");
87      Real KvI;
88      Real vI;
89      Real zetaI;
```

```
90        Integer index;
91        Real Q(unit = "kg/h");
92        Real Kvs = 0.86 * Cv;
93        constant Real PI = 3.1415 "9265358979";
94        constant Real rho100 = 1000;
95        constant Real dP100 = 100000;
96        Real mVap;
97        Boolean run;
98        Real inputRegion,outputRegion;
99        Real X(unit = "[%]/100");
100       Real Tsaturated,hLiquidOut,hVapourOut,hInput;
101       Real HsumInput,HsumOutput;
102       Real Qpercent,vZeta;
103       parameter Real dQ[1,12];
104       parameter Real minPout;
105       Models.BackPressureConnector pressurePort;
106       Real Kv[1,12] = Kvs * dQ[:] / 100;
107     protected
108       Real unlimiteddP,tempDPMax;
109     algorithm
110       if hub <= 0 then
111         index:=1;
112       elseif hub <= 10 then
113         index:=2;
114       elseif hub < 20 then
115         index:=3;
116       elseif hub <= 30 then
117         index:=4;
118       elseif hub <= 40 then
119         index:=5;
120       elseif hub <= 50 then
121         index:=6;
122       elseif hub <= 60 then
123         index:=7;
124       elseif hub <= 70 then
125         index:=8;
126       elseif hub <= 80 then
127         index:=9;
128       elseif hub <= 90 then
129         index:=10;
130       else
131         index:=12;
132       end if;
133     equation
134       if index > 1 and run then
135         vZeta = KvI * 4 / (D * D * PI * 3600);
136         zetaI = dP100 * 2 / (rho100 * vZeta * vZeta);
137         KvI = Kvs * Qpercent;
138         vI = 4 * Q / (rho * 3600 * D ^ 2 * PI);
139         unlimiteddP = -rho * vI * vI * zetaI / 200000;
140       elseif index > 1 and run == false then
141         vZeta = KvI * 4 / (D * D * PI * 3600);
142         zetaI = dP100 * 2 / (rho100 * vZeta * vZeta);
143         KvI = Kvs;
144         vI = 4 * Q / (rho * 3600 * D ^ 2 * PI);
145         unlimiteddP = -rho * vI * vI * zetaI / 200000;
146       else
147         vZeta = 0;
148         zetaI = 0;
149         c = 9;
150         KvI = 0;
151         vI = 0;
152         unlimiteddP = inputPort.P;
153       end if;
154       Q = inputPort.m * timeFactor;
155       inputRegion = Functions.region_PT(inputPort.P, inputPort.T);
156       outputRegion = Functions.region_PT(outletPort.P, inputPort.T);
157       if outputRegion > 1 and outputRegion < 3 then
158         dmV = mVap;
159         dm = -mVap;
160         mVap = -inputPort.m * (1 - X) * 1;
161       elseif outputRegion > 0 then
162         dmV = 0;
163         dm = 0;
164         mVap = 0;
165       else
166         dmV = 0;
```

```
167         dm = 0;
168         mVap = 0;
169       end if;
170       if hLiquidOut > 0 and hVapourOut > 0 and hLiquidOut <> hVapourOut then
171         X = min(max(0.0, (hInput - hVapourOut) / (hLiquidOut - hVapourOut)), 1.0);
172       else
173         X = 0;
174       end if;
175       if Tsaturated <= inputPort.T then
176         dT = Tsaturated - inputPort.T;
177       else
178         dT = 0;
179       end if;
180       tempDPMax = min(-0.1, minPout - inputPort.P);
181       dP = max(unlimiteddP, tempDPMax);
182       hInput = Functions.h_pT(inputPort.P, inputPort.T);
183       hLiquidOut = Functions.hL_p(outletPort.P);
184       hVapourOut = Functions.hV_p(outletPort.P);
185       Tsaturated = Functions.Tsat_p(outletPort.P);
186       HsumInput = inputPort.m * hInput;
187       HsumOutput = inputPort.m * X * hLiquidOut + inputPort.m * (1 - X) * hVapourOut;
188       pressurePort.dP = dP;
189     end Valve;
190     partial model Separator
191       parameter Real timeFactor;
192       parameter Real startPressure "unit = [bar]";
193       parameter Real startTemperature "unit = [C]";
194       parameter Real startVolume "unit = [m3]";
195       Real parameterMaxVol "unit = [m3]";
196       parameter Real rho = 1000 "unit = kg/m3";
197       parameter Real openingArea = 0.05 * 0.05 * 3.14 / 4;
198       parameter Real height;
199       parameter Real area "unit=[m]";
200       parameter Real startq;
201       Boolean selfFlow;
202       Real Pressure;
203       Real Temperature;
204       Real Mass;
205       Real Volume;
206       discrete Real q(start = startq);
207       Models.StreamConnector outletPort2;
208       Models.StreamConnector inputPort2;
209       Models.StreamConnector inputPort;
210       Models.StreamConnector outletPort;
211     protected
212       Real mDrawn;
213       Real MassVap;
214       Real heigthDiff;
215       Real derMass,derPressure;
216       Real TemperatureNew;
217       Real vv "v2 m/s";
218       Real v;
219       Real Q "unit = kg/h";
220       Real inputM1,inputMV1,inputM2,inputMV2,outletM1,outletMV1,outletM2,outletMV2;
221     equation
222       outletPort.P = Pressure;
223       parameterMaxVol = height * area;
224       outletPort.mV = 0;
225       outletPort.T = Temperature;
226       heigthDiff = Volume / parameterMaxVol * height;
227       vv = 2 * 9.81 * heigthDiff;
228       v = sqrt(max(0, vv));
229       when sample(1, 5) then
230             q = -v * rho * 3600 * openingArea;
231       end when;
232       Q = min(q, mDrawn);
233       outletPort.m = Q;
234       outletPort2.mV + inputPort.mV = 0;
235       outletPort2.T = Temperature;
236       outletPort2.P = Pressure;
237       outletPort2.m = 0;
238       derMass = inputM1 + inputM2 + inputMV1 + inputMV2 + outletM1 + outletM2 + outletMV1 + outletMV2;
239       der(Mass) = Functions.Utility.Range(Mass, 0, parameterMaxVol * rho, derMass);
240       MassVap = 0;
241       Volume = Mass / rho;
242       TemperatureNew = Functions.T_misch(inputPort.T, inputPort2.T, pre(Temperature),
243                 inputM1, inputMV1, inputM2, inputMV2, pre(Mass), pre(MassVap), outletM1, outletMV2);
```

```
244       der(Temperature) = (-Temperature + TemperatureNew) * 3600 / timeFactor;
245       derPressure = 0;
246       der(Pressure) = derPressure;
247       inputM1 = inputPort.m / 3600 * timeFactor;
248       inputMV1 = inputPort.mV / 3600 * timeFactor;
249       inputM2 = inputPort2.m / 3600 * timeFactor;
250       inputMV2 = inputPort2.mV / 3600 * timeFactor;
251       outletM1 = outletPort.m / 3600 * timeFactor;
252       outletMV1 = outletPort.mV / 3600 * timeFactor;
253       outletM2 = outletPort2.m / 3600 * timeFactor;
254       outletMV2 = outletPort2.mV / 3600 * timeFactor;
255     end Separator;
256     connector BackPressureConnector
257       Real dP;
258       flow Real flow_;
259     end BackPressureConnector;
260     model BackPressureSource
261       Models.BackPressureConnector outletPort;
262       Real dP;
263     equation
264       outletPort.dP = dP;
265       outletPort.flow_ = 0;
266     end BackPressureSource;
267 end Models;
```

Code A.3: Die speziellen Modelle.

```
1   package SpecificModels
2     model ValveCV15
3       extends Models.Valve(rho = 1000, Cv = 15, dQ = [0.0,4,7,10,15,19,28,38,50,70,88,100],
4             D = 0.025, timeFactor = timeFactor);
5     end ValveCV15;
6     model WEPA_HeatExchanger
7       extends Models.HeatExchanger;
8     end WEPA_HeatExchanger;
9     model WEPA_ZP
10      extends Models.Pump;
11      Real mDrawn;
12      Real testing;
13      parameter Real timeFactor;
14      Boolean run;
15    equation
16      X2 = dPback / Vold ^ 2;
17      if rpmPercentage < 70 then
18        testing = 69;
19        Pnew = 3.39055 * 10 ^ (-7) * Vnew ^ 4 + 0.0000331349 * Vnew ^ 3 - 0.0207107 * Vnew ^ 2
20            + 0.144132 * Vnew + 0.0121305 * rpmPercentage ^ 2 - 0.00263577 * rpmPercentage + 0.577019;
21      elseif rpmPercentage < 90 then
22        testing = 89;
23        Pnew = 0.0000125619 * (-1) * Vnew ^ 4 + 0.000963911 * Vnew ^ 3 - 0.0416786 * Vnew ^ 2
24            + 0.431428 * Vnew + 0.0121305 * rpmPercentage ^ 2 - 0.00263577 * rpmPercentage + 0.577019;
25      else
26        testing = 100;
27        Pnew = (-0.0233012) * Vnew ^ 2 + 0.490815 * Vnew + 0.0121305 * rpmPercentage ^ 2
28            - 0.00263577 * rpmPercentage + 0.577019;
29      end if;
30      Vmax = 1.82853 * 10 ^ (-6) * rpmPercentage ^ 4 - 0.000397507 * rpmPercentage ^ 3
31            + 0.0291172 * rpmPercentage ^ 2 - 0.278489 * rpmPercentage + 9.26867;
32      Vnew = min(Vmax, sqrt(Pnew / X2));
33      Vold = inputPort.m / rho * timeFactor;
34      dT = 0;
35      if run then
36        mDrawn = -pre(Vnew) * rho / timeFactor;
37        dP = Pnew / 10;
38      else
39        mDrawn = -pre(inputPort.m);
40        dP = 0;
41      end if;
42      annotation(experiment(StartTime = 0, StopTime = 2000, Tolerance = 0.01));
43    end WEPA_ZP;
44    model ValveCV180
45      extends Models.Valve(rho = 1000, Cv = 180, dQ = [0.001,1,4,8,10,16,21,30,40,60,80,100],
46            D = 0.05, timeFactor = timeFactor);
```

```
    equation
      Qpercent = (4.36427 * 10 ^ (-6) * hub ^ 4 - 0.000638956 * hub ^ 3 + 0.0330616 * hub ^ 2
            - 0.290693 * hub + 0.486376) / 100;
    end ValveCV180;
    model WEPA_Seperator
      extends Models.Seperator;
    initial equation
      Volume = startVolume;
      Temperature = startTemperature;
      Pressure = startPressure;
    end WEPA_Seperator;
  end SpecificModels;
```

Code A.4: Die eigenen Funktionen.

```
package Functions
  function v2_PT
    input Real p;
    input Real T;
    output Real v2_pT;
  protected
    Real i;
    Real tau,go_pi,gr_pi;
    Real Ir[1,:] = [1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,5,6,6,6,7,7,7,8,8,9,10,10,10,16,16,
          18,20,20,20,21,22,23,24,24,24];
    Real Jr[1,:] = [0,1,2,3,6,1,2,4,7,36,0,1,3,6,35,1,2,3,7,3,16,35,0,11,25,8,36,13,4,10,
          14,29,50,57,20,35,48,21,53,39,26,40,58];
    Real nr[1,:] = [-0.0017731742473213,-0.017834862292358,-0.045996013696365,-0.057581259083432,
          -0.05032527872793,-0.00003303264167020 3,-0.00018948987516315,-0.0039392777243355,
          -0.043797295650573,-0.000026674547914087,0.00000020481737692309,0.0000004387066728 4435,
          -0.00003227767723857,-0.0015033924542148,-0.040668253562649,-0.00000000078847309559367,
          0.00000001279071785 2285,0.0000004822537271 8507,0.0000022922076337661,-0.00000000001671476645 1061,
          -0.00211714723213 55,-23.895741934104,-0.00000000000000000059059564324 27,-0.0000012621808899101,
          -0.038946842435739,0.00000000001125621136045 9,-8.2311340897998,0.0000000198097128020 88,
          0.00000000000000000010406965210174,-0.00000000000010234747095929,-0.0000000010018179379511,
          -0.00000000008088290864698 5,0.10693031879409,-0.33662250574171,0.00000000000000000000000089185845355421,
          0.00000000000030629316876232,-0.000004200246769820 8,-0.00000000000000000000000059056029685639,
          0.00000378269476134 57,-0.00000000000000012768608934681,0.00000000000000000000000000073087610595061,
          0.00000000000000005541471535077 8,-0.0000009436970724121];
    Real Jo[1,:] = [0,1,-5,-4,-3,-2,-1,2,3];
    Real no[1,:] = [-9.6927686500217,10.086655968018,-0.005608791128302,0.071452738081455,
          -0.40710498223928,1.4240819171444,-4.383951131945,-0.28408632460772,0.021268463753307];
    constant Real R = 0.461526 "kJ/(kg K)";
  algorithm
    tau:=540 / T;
    go_pi:=1 / p;
    gr_pi:=0;
    for i in 1:43 loop
        gr_pi:=gr_pi + nr[1,i] * Ir[1,i] * p ^ (Ir[1,i] - 1) * (tau - 0.5) ^ Jr[1,i];
    end for;
    v2_pT:=R * T / p * p * (go_pi + gr_pi) / 1000;
  end v2_PT;
  function v1_pT
    input Real p;
    input Real T;
    output Real v1_pT_;
  protected
    constant Real R = 0.461526 "kJ/(kg K)";
    Real ps;
    Real tau;
    Real g_p;
    Real I1[1,:] = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,3,3,3,4,4,4,5,8,8,21,23,29,30,31,32];
    Integer J1[1,:] = [-2,-1,0,1,2,3,4,5,-9,-7,-1,0,1,3,-3,0,1,3,17,-4,0,6,-5,-2,10,-8,-11,-6,
          -29,-31,-38,-39,-40,-41];
    Real n1[1,:] = [0.14632971213167,-0.84548187169114,-3.756360367204,3.3855169168385,
          -0.95791963387872,0.15772038513228,-0.016616417199501,0.00081214629983568,0.00028319080123804,
          -0.00060706301565874,-0.018990068218419,-0.032529748770505,-0.021841717175414,-0.00005283835796993,
          -0.00047184321073267,-0.00030001780793026,0.00004766139390698 7,-0.0000044141845330846,
          -0.00000000000000072694996297594,-0.000031679644845054,-0.000002827079798 5312,-0.00000000085205128120103,
          -0.0000022425281908,-0.000000651712228956 01,-0.00000000000014341729937924,-0.0000004051699686011 7,
          -0.00000001273430174 1641,-0.00000000017424871230634,-0.00000000000000000068762131295531,
          0.00000000000000000001447830782852 1,0.00000000000000000000026335781662795,
          -0.00000000000000000000011947622640071,0.00000000000000000000001822809458 1404,
```

```
59            -0.00000000000000000000000000935370872924 58];
60      algorithm
61        ps:=p / 16.53;
62        tau:=1386 / T;
63        g_p:=0;
64        for i in 1:34 loop
65              g_p:=g_p - n1[1,i] * I1[1,i] * (7.1 - ps) ^ (I1[1,i] - 1) * (tau - 1.222) ^ J1[1,i];
66        end for;
67        v1_pT_:=R * T / p * ps * g_p / 1000;
68      end v1_pT;
69      function v_pT
70        input Real P;
71        input Real t;
72        output Real v_pT;
73      protected
74        Real p,T;
75        Integer region;
76        constant Real xlErrValue = 0;
77      algorithm
78        p:=P / 10;
79        T:=t + 273.15;
80        region:=region_PT(p, T);
81        if region == 1 then
82          v_pT:=v1_pT(p, T);
83        elseif region == 2 then
84          v_pT:=v2_PT(p, T);
85        else
86          v_pT:=xlErrValue;
87        end if;
88      end v_pT;
89      function Tsat_p
90        input Real P;
91        output Real Tsat_p;
92      protected
93        Real p;
94        constant Real xlErrValue = 0;
95      algorithm
96        p:=P / 10;
97        if p >= 0.000611657 and p <= 22.06395 + 0.001 then
98          Tsat_p:=T4_p(p) - 273.15;
99        else
100         Tsat_p:=xlErrValue;
101       end if;
102     end Tsat_p;
103     function T4_p
104       input Real p;
105       output Real T4_p;
106     protected
107       Real beta,e,f,g,d;
108     algorithm
109       beta:=p ^ 0.25;
110       e:=beta ^ 2 - 17.073846940092 * beta + 14.91510861353;
111       f:=1167.0521452767 * beta ^ 2 + 12020.82470247 * beta - 4823.2657361591;
112       g:=-724213.16703206 * beta ^ 2 - 3232555.0322333 * beta + 405113.40542057;
113       d:=2 * g / (-f - (f ^ 2 - 4 * e * g) ^ 0.5);
114       T4_p:=(650.17534844798+d-((650.17534844798 + d)^2 - 4*(-0.23855557567849 + 650.17534844798 * d))^ 0.5)/2;
115     end T4_p;
116     function T_misch
117       input Real Tin;
118       input Real Tin2;
119       input Real Tis;
120       input Real min;
121       input Real minV;
122       input Real min2;
123       input Real min2V;
124       input Real mold;
125       input Real moldV;
126       input Real mout;
127       input Real moutV;
128       output Real T;
129     algorithm
130       T:=(Tin * min + Tin * minV + Tin2 * min2 + Tin2 * min2V + Tis * (mold + mout) + Tis * (moldV + moutV))
131             / (min + minV + min2 + min2V + mold + mout + moldV + moutV);
132     end T_misch;
133     function rho_pT
134       input Real p;
135       input Real T;
```

```
136       output Real rho_pT;
137     algorithm
138       rho_pT:=1 / v_pT(p, T);
139     end rho_pT;
140     function p3sat_h
141       input Real hin;
142       output Real p3sat_h;
143     protected
144       Real Ii[1,:] = [0,1,1,1,1,5,7,8,14,20,22,24,28,36];
145       Real Ji[1,:] = [0,1,3,4,36,3,0,24,16,16,3,18,8,24];
146       Real ni[1,:] = [0.600073641753024,-9.36203654849857,24.6590798594147,-107.014222858224,
147         -91582131580576.8,-8623.32011700662,-23.5837344740032,2.52304969384128e+17,-3.897187719977719e+18,
148         -3.33775713645296e+22,35649946963.6328,-1.485475447206 41e+26,3.3061151483879 8e+18,8.136412944678 29e+37];
149       Real ps,h;
150       Integer i;
151     algorithm
152       h:=hin / 2600;
153       ps:=0;
154       for i in 1:14 loop
155             ps:=ps + ni[1,i] * (h - 1.02) ^ Ii[1,i] * (h - 0.608) ^ Ji[1,i];
156       end for;
157       p3sat_h:=ps * 22;
158     end p3sat_h;
159     function hV_p
160       input Real P;
161       output Real hV_p;
162     protected
163       Real p,Low_Bound,High_Bound,hs,ps,Ts;
164       constant Real xlErrValue = 0;
165     algorithm
166       p:=P / 10;
167       if p > 0.000611657 and p < 22.06395 then
168         if p > 0.000611657 and p < 22.06395 then
169           Ts:=T4_p(p);
170           if p < 16.529 then
171             hV_p:=h2_pT(p, Ts);
172           else
173             Low_Bound:=2087.23500164864;
174             High_Bound:=2563.592004 + 5;
175             while (abs(p - ps) > 0.000001) loop
176                           hs:=(Low_Bound + High_Bound) / 2;
177               ps:=p3sat_h(hs);
178               if ps < p then
179                 High_Bound:=hs;
180               else
181                 Low_Bound:=hs;
182               end if;
183             end while;
184             hV_p:=hs;
185           end if;
186         else
187           hV_p:=xlErrValue;
188         end if;
189       else
190         hV_p:=xlErrValue;
191       end if;
192     end hV_p;
193     function hL_p
194       input Real P;
195       output Real h4L_p;
196     protected
197       Real p,Low_Bound,High_Bound,hs,ps,Ts;
198       constant Real xlErrValue = 0;
199     algorithm
200       p:=P / 10;
201       if p > 0.000611657 and p < 22.06395 then
202         if p > 0.000611657 and p < 22.06395 then
203           Ts:=T4_p(p);
204           if p < 16.529 then
205             h4L_p:=h1_pT(p, Ts);
206           else
207             Low_Bound:=1670.858218;
208             High_Bound:=2087.23500164864;
209             while (abs(p - ps) > 0.00001) loop
210                           hs:=(Low_Bound + High_Bound) / 2;
211               ps:=p3sat_h(hs);
212               if ps > p then
```

```
213                 High_Bound:=hs;
214               else
215                 Low_Bound:=hs;
216               end if;
217             end while;
218             h4L_p:=hs;
219           end if;
220         else
221           h4L_p:=xlErrValue;
222         end if;
223       else
224         h4L_p:=xlErrValue;
225       end if;
226     end hL_p;
227     function h2_pT
228       input Real p;
229       input Real T;
230       output Real h2_pT;
231     protected
232       Integer i;
233       Real Ir[1,:] = [1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,5,6,6,6,7,7,7,8,8,9,10,10,10,16,16,
234             18,20,20,20,21,22,23,24,24,24];
235       Real Jr[1,:] = [0,1,2,3,6,1,2,4,7,36,0,1,3,6,35,1,2,3,7,3,16,35,0,11,25,8,36,13,4,10,14,
236             29,50,57,20,35,48,21,53,39,26,40,58];
237       Real nr[1,:] = [-0.0017731742473213,-0.017834862292358,-0.045996013696365,-0.057581259083432,
238             -0.05032527872793,-0.000033032641670203,-0.00018948987516315,-0.0039392777243355,
239             -0.043797295650573,-0.00002667454791408 7,0.000000020481737692309,0.0000004387066728 4435,
240             -0.00003227767723857,-0.0015033924542148,-0.040668253562649,-0.000000000788473095593 67,
241             0.000000012790717852285,0.000000048225372718507,0.00000229220763376 61,-0.000000000001671476645106 1,
242             -0.0021171472321355,-23.895741934104,-0.000000000000000005905956432427,-0.000001262180889910 1,
243             -0.038946842435739,0.000000000011256211360459,-8.2311340897998,0.000000019809712802088,
244             0.000000000000000000010406965210174,-0.000000000000010234747095929,-0.000000001000181793795 11,
245             -0.000000000080882908646985,0.10693031879409,-0.33662250574171,0.000000000000000000000000000089185845355421,
246             0.000000000000030629316876232,-0.000004200246769820 8,-0.000000000000000000000000059056029685639,
247             0.0000037826947613457,-0.000000000000001276860893468 1,0.000000000000000000000000000073087610595061,
248             0.000000000000000055414715350778,-0.0000009436970724121];
249       Real Jo[1,:] = [0,1,-5,-4,-3,-2,-1,2,3];
250       Real no[1,:] = [-9.6927686500217,10.086655968018,-0.005608791128302,0.071452738081455,
251             -0.40710498223928,1.4240819171444,-4.383951131945,-0.28408632460772,0.021268463753307];
252       Real tau,go_tau,gr_tau;
253       constant Real R = 0.461526 "kJ/(kg K)";
254     algorithm
255       tau:=540 / T;
256       go_tau:=0;
257       for i in 1:9 loop
258           go_tau:=go_tau + no[1,i] * Jo[1,i] * tau ^ (Jo[1,i] - 1);
259       end for;
260       gr_tau:=0;
261       for i in 1:43 loop
262           gr_tau:=gr_tau + nr[1,i] * p ^ Ir[1,i] * Jr[1,i] * (tau - 0.5) ^ (Jr[1,i] - 1);
263       end for;
264       h2_pT:=R * T * tau * (go_tau + gr_tau);
265     end h2_pT;
266     function h1_pT
267       input Real P;
268       input Real T;
269       output Real h1_pT;
270     protected
271       Integer i;
272       Real ps,tau,g_t,p;
273       Real I1[1,:] = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,3,3,3,4,4,4,5,8,8,21,23,29,30,31,32];
274       Real J1[1,:] = [-2,-1,0,1,2,3,4,5,-9,-7,-1,0,1,3,-3,0,1,3,17,-4,0,6,-5,-2,10,-8,-11,-6,
275             -29,-31,-38,-39,-40,-41];
276       Real n1[1,:] = [0.14632971213167,-0.84548187169114,-3.756360367204,3.38551691683 85,
277             -0.95791963387872,0.15772038513228,-0.016616417199501,0.00081214629983568,
278             0.00028319080123804,-0.00060706301565874,-0.018990068218419,-0.032529748770505,
279             -0.021841717175414,-0.00005283835796993,-0.00047184321073267,-0.00030001780793026,
280             0.00004766139390 6987,-0.00000441418453308 46,-0.000000000000007269499629 7594,
281             -0.00003167964484505 4,-0.00000282707979853 12,-0.000000000852051281201 03,
282             -0.000002242528190 8,-0.00000065171222895601,-0.000000000000143417299379 24,
283             -0.00000040516996860117,-0.000000001273430174 1641,-0.000000000174248712306 34,
284             -0.000000000000000006876213129553 1,0.000000000000000000014478307828521,
285             0.000000000000000000000026335781662795,-0.000000000000000000000011947622640071,
286             0.000000000000000000000018228094581404,-0.000000000000000000000000935370872924 58];
287       constant Real R = 0.461526 "kJ/(kg K)";
288     algorithm
289       p:=P / 16.53;
```

```
290       tau:=1386 / T;
291       g_t:=0;
292       for i in 1:34 loop
293             g_t:=g_t + n1[1,i] * (7.1 - p) ^ I1[1,i] * J1[1,i] * (tau - 1.222) ^ (J1[1,i] - 1);
294       end for;
295       h1_pT:=R * T * tau * g_t;
296     end h1_pT;
297     function h_pT
298       input Real P;
299       input Real t;
300       output Real h_pT;
301     protected
302       Real p,T,region;
303       constant Real xlErrValue = 0;
304     algorithm
305       p:=P / 10;
306       T:=t + 273.15;
307       region:=region_PT(P, t);
308       if region == 1 then
309         h_pT:=h1_pT(p, T);
310       elseif region == 2 then
311         h_pT:=h2_pT(p, T);
312
313       elseif region == 3 then
314         h_pT:=xlErrValue;
315
316       elseif region == 4 then
317         h_pT:=xlErrValue;
318
319       elseif region == 5 then
320         h_pT:=xlErrValue;
321       else
322         h_pT:=xlErrValue;
323       end if;
324     end h_pT;
325     function region_PT
326       input Real Pin;
327       input Real Tin;
328       output Integer region;
329     protected
330       Real ps,P,T;
331       Real B23p_T,p4_T;
332       Real teta,a,b,c;
333     algorithm
334       P:=Pin / 10;
335       T:=Tin + 273.15;
336       B23p_T:=348.05185628969 - 1.1671859879975 * T + 0.0010192970039326 * T ^ 2;
337       teta:=T - 0.23855557567849 / (T - 650.17534844798);
338       a:=teta ^ 2 + 1167.0521452767 * teta - 724213.16703206;
339       b:=-17.073846940092 * teta ^ 2 + 12020.82470247 * teta - 3232555.0322333;
340       c:=14.91510861353 * teta ^ 2 - 4823.2657361591 * teta + 405113.40542057;
341       p4_T:=(2 * c / (-b + (b ^ 2 - 4 * a * c) ^ 0.5)) ^ 4;
342       if T > 1073.15 and P < 10 and T < 2273.15 and P > 0.000611 then
343         region:=5;
344       elseif T <= 1073.15 and T > 273.15 and P <= 100 and P > 0.000611 then
345         if T > 623.15 then
346           if P > B23p_T then
347             region:=3;
348             if T < 647.096 then
349               ps:=p4_T;
350               if abs(P - ps) < 0.00001 then
351                 region:=4;
352               else
353
354               end if;
355             else
356
357             end if;
358           else
359             region:=2;
360           end if;
361         else
362           ps:=p4_T;
363           if abs(P - ps) < 0.00001 then
364             region:=4;
365           elseif P > ps then
366             region:=1;
```

```
          else
            region:=2;
          end if;
        end if;
      else
        region:=0;
      end if;
    end region_PT;
    package Utility
      function Range
        input Real value;
        input Real min;
        input Real max;
        input Real derval;
        output Real newDer;
      algorithm
        if value >= max and derval > 0 then
          newDer:=0;
        elseif value < min and derval < 0 then
          newDer:=0;
        else
          newDer:=derval;
        end if;
      end Range;
    end Utility;
  end Functions;
```

Code A.5: Die Regler.

```
package Controller
    function ControllerPump
      input Real P;
      input Real targetP;
      input Real Phub;
      input Real thresh;
      input Real stepSize;
      output Real dPhub;
    protected
      Real dhub;
    algorithm
      if P > targetP + thresh then
        dhub:=-stepSize;
      elseif P < targetP - thresh then
        dhub:=stepSize;
      else
        dhub:=0;
      end if;
      if Phub >= 100 and dhub > 0 then
        dPhub:=0;
      elseif Phub <= 0 and dhub < 0 then
        dPhub:=0;
      else
        dPhub:=dhub;
      end if;
    end ControllerPump;
    function ControllerValve
      input Real T;
      input Real P;
      input Real Tthresh;
      input Real dPOld;
      input Real dTMax;
      input Real dTMin;
      output Real dhub;
      output Real state;
    protected
      Real Tsat;
      Real TAfter;
      Real dHubSimple;
    algorithm
      Tsat:=Functions.Tsat_p(P);
      TAfter:=Functions.Tsat_p(P + dPOld);
      if Tsat - Tthresh > T then
        dHubSimple:=-dTMin;
```

```
45          state:=1;
46        elseif T > Tsat - Tthresh / 2 then
47          dHubSimple:=dTMax;
48          state:=2;
49
50        elseif T > Tsat - Tthresh then
51          dHubSimple:=dTMin;
52          state:=3;
53
54        elseif Tsat - Tthresh * 1.1 > T and T > TAfter then
55          dHubSimple:=dTMin / 2;
56          state:=4;
57
58        elseif Tsat - Tthresh > T and T > TAfter then
59          dHubSimple:=0;
60          state:=5;
61        else
62          dHubSimple:=0;
63          state:=6;
64        end if;
65        dhub:=dHubSimple;
66      end ControllerValve;
67      function ControllerHeatexchanger
68        extends ControllerDoubleStep;
69      end ControllerHeatexchanger;
70      partial function ControllerDoubleStep
71        input Real is;
72        input Real target;
73        input Real bigStep;
74        input Real smallStep;
75        input Real rangeLow;
76        input Real rangeHigh;
77        input Real thresh;
78        input Real oldValue;
79        output Real result;
80      protected
81        Real hub;
82      algorithm
83        if is < target + 2 * thresh then
84          hub:=bigStep;
85        elseif is < target + thresh then
86          hub:=smallStep;
87
88        elseif is > target + thresh then
89          hub:=-smallStep;
90
91        elseif is > target + thresh / 2 then
92          hub:=-bigStep;
93        else
94          hub:=0;
95        end if;
96        if oldValue + hub > rangeHigh then
97          result:=0;
98        elseif oldValue + hub < rangeLow then
99          result:=0;
100       else
101         result:=hub;
102       end if;
103     end ControllerDoubleStep;
104 end Controller;
```