On Model Libraries for Thermo-hydraulic Applications

Eborn, Jonas

2001

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Eborn, J. (2001). *On Model Libraries for Thermo-hydraulic Applications*. [Doctoral Thesis (compilation), Department of Automatic Control]. Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

On Model Libraries for Thermo-hydraulic Applications

Jonas Eborn

Automatic Control

On Model Libraries for Thermo-hydraulic Applications

On Model Libraries for Thermo-hydraulic Applications

Jonas Eborn

Department of Automatic Control
Lund Institute of Technology
Lund, March 2001

Tillägnas min familj

Pernilla, Elise & Per

Dedicated to my family

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT–1061–SE

Printed in Sweden by Bloms i Lund Tryckeri AB.
Lund 2001

Contents

List of Symbols

Symbol	In 4, IV	Unit	Description/Quantity
	α_r	1	Steam mass fraction
	α_v	1	Steam volume fraction
ΔT_{lm}		K	Logarithmic mean temperature diff.
Δz		m	Discretization length
	δl	mm	Water level
κ		1	Ratio of specific heats
ν		m^3/kg	Volumity
ξ		m	Position
ρ		kg/m^3	Density
	σ	–	Output variance
	τ	s	Time constant
A		m^2	Area
a_r		1	Steam mass fraction
C_p		J/kgK	Specific heat capacity
E		J	Total energy
F		N	Force
g		m/s^2	Gravitational constant, 9.81
h		J/kg	Specific enthalpy
h_c		J/kg	Evaporation enthalpy
I		kgm/s	Momentum
G		kg/sm	Momentum flux
k		1	Overall friction coefficient
k		N/m	Spring constant
l, L		m	Length
$\mathcal{M}$		–	Model structure
m, M		kg	Mass
$\dot{m}$		kg/s	Mass flow rate
$\dot{m}_x$		kg/s	Component mass flow rate
n		rpm	Rotational speed
n		1	Discretization number
p		Pa	Pressure

Symbol	In 4, IV	Unit	Description/Quantity
P		J/sm	Heat flow per unit length
R		m^{2}K/W	Heat resistance
	q	kg/s	Mass flow
q	Q	J/s	Heat flow
q_c		J/s	Convective heat flow
T		K	Temperature
$\mathbf{T}$		N/m^2	Stress tensor
u		J/kg	Specific internal energy
U		J	Total internal energy
V		m^3	Volume
w		m/s	Flow velocity
x		m	position
x		1	mass flow ratio in Paper III
	x_r	1	Mass fraction

Acknowledgements

First of all I would like to thank my supervisor Professor Emeritus[1] Karl Johan Åström for building a department with such a creative atmosphere. It has been a great pleasure and privilege to work with you. I also wish to thank my two other supervisors Dr Sven Erik Mattsson (95–98) and Professor Anders Rantzer (98–01) for always having faith in me. I am very grateful for the work they and other senior staff at the department do to acquire funding for many graduate students.

This thesis is based on collaborations with many people, who besides being colleagues also became wonderful friends. Thanks to Bernt Nilsson who got me started on **K2** and gave me a lot of insight. Many thanks to Hubertus Tummescheit, who got me restarted after I finished my licentiate thesis and provided a lot of domain knowledge for the design of ThermoFlow. I also wish to thank James Sørlie and Falko Jens Wagner whom I have had the pleasure of working with.

Special thanks go to the people at Dynasim AB, who kept us updated with new Dymola versions and helped solving the endless line of software problems that was triggered by the library design. Many thanks also to those who proofread parts of my thesis, Anders Robertsson and Jakob Munch Jensen among others.

I also want to thank all the people at the department who all have a part in making it such a wonderful place: Leif and Anders for keeping the computer system in shape; Eva, Bittan and Agneta for keeping our spirits up and feet down on the ground; Anton, Ari, Sven and Andrey for their extra-curricular activities such as poker, malt whisky, Skrylle and sauna.

This project has jointly been supported by Sydkraft AB under the project name ”Modelling and Control of Energy Systems” and by the National Board for Industrial and Technical Development (NUTEK) programme ”Complex Systems”, Dnr 96-10653. Their financial support is gratefully acknowledged. I also wish to extend my gratitude to people at Sycon AB (formerly Sydkraft Konsult) who have collaborated with us in the project, especially to Jan Tuszyński and Jörgen Svensson.

Finally, I want to thank my family and friends for making me the person I am today. Per, Elise and Pernilla, you are all the greatest thing that has happened to me. I love you for bringing me so much joy.

Jonas

[1] As of January 2000.

A note on language

The papers in this thesis were written in cooperation with several coauthors. Because of this there is both American and British spelling used in the different papers, e. g., the word *modelling* becomes with American spelling *modeling*. I am aware of this and I hope that it does not bother the reader too much.

Note also that in the introductory chapters I have referenced figures in the papers using the notation II.4 although the figure in Paper II is only labelled Figure 4.

1

Introduction

Abstract

This introductory chapter gives some motivation for the work in this thesis and explains how this piece fits into the jigsaw puzzle of automatic control.

1.1 Motivation

Today control appears in almost every technical system in all possible engineering domains. Some kind of model of the real world system is always needed to design and implement a control system. Modelling thus provides a bridge between the real world and the automatic control world,

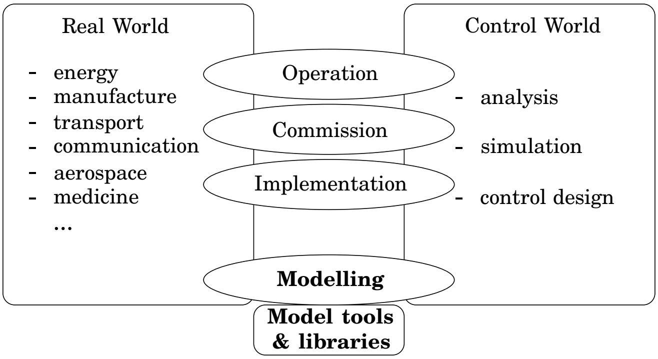


Figure 1.1 Modelling bridges the gap between the real world and control.

see Figure 1.1. Typically, a mathematical model is needed in all different aspects of the work done in automatic control; analysis, control design and simulation.

Why modelling?

A good model provides knowledge of a system. A model of the process can give increased understanding, and with that comes also better possibilities to increase quality, safety and economy. This has been very well expressed by an executive at one of the largest companies in the process industry:

> Modeling and simulation technologies are keys to achieve *manufacturing excellence* and to *assess risk* in unit operations. As we make our plant more flexible to respond to *business opportunities*, efficient *modeling and simulation* techniques will become *commonly used tools*.
>
> Ralph P. Schlenker, Exxon Chemical

There is a need for tools that can provide insight into complex systems. Technical systems are becoming increasingly more complex mainly for two reasons:

Integration of systems introduces tight couplings where previously parts could be designed and operated independently. Integration comes from recirculation of materials to reduce energy consumption and pollution. It also comes from reduction or removal of buffers to reduce production times and increase the flexibility of the plant, as stated in the quote above.

Heterogeneity in systems forces a mixture of several engineering disciplines to be considered simultaneously. Heterogeneity is of course a consequence of integration, for example in mechatronic systems where mechanics, electronics and control algorithms interact closely, but it is also an additional difficulty, adding complexity. Different traditions of how to treat systems give rise to, for example, mixtures of continuous time and discrete time systems.

Mathematical modelling is a fundamental tool to tackle complex systems. The main use of models of complex systems is today in simulation. Simulation is important since it provides the possibility to study the behaviour of a model and draw conclusions concerning the real world system. Simulation has been the main tool to verify the demands that should be achieved by a control system, e. g.,

- Environmental demands
- Safety demands

- Economical demands
- Quality/performance demands

The focus that for a long time has been on simulation is now changing towards analysis. Tools that from models of complex systems can extract simpler models useful for analysis is still lacking though. This is, however, outside the scope of this thesis, but some comments on such tools are given in Chapter 6.

Models are also starting to appear as explicit components in control systems. Typical examples are in observers for variables that are not measured directly and in model predictive controllers, see Maciejowski (2000).

Different fields – different traditions

In different areas of engineering there have been very different traditions in what kind of models and how modelling has been used. Some examples are:

Automatic Control – Block diagram modelling, either with transfer functions or with state-space models.

Circuit Simulation – Signal flow modelling, large nets with many similar components.

Chemical Processes – Static design calculations, using flow-sheeting to build process diagrams.

Mechanical Engineering – CAD tools for multi-body systems, with much emphasis on the visual appearance.

Power Systems – Special purpose tools, either for static design calculations of heat balances, or for power grid simulations, usually dynamic.

The main topic of this thesis, physical modelling, can form a basis in any of the modelling methods mentioned above. In automatic control for example, physical modelling coexists with system identification methods, which is another way to find models. In the chemical process industry physical modelling is the most natural approach, but there the focus has traditionally been on static mass and energy balances and not on dynamic properties which are most important for control purposes.

Why model libraries?

The scope of this thesis is mainly on how to build and use model libraries for thermal power systems. Providing model libraries is an excellent way to package modelling knowledge that can help others with similar problems. Good model libraries are often the primary reason for the use of special purpose simulation software, like Spice and Saber for electrical circuits, Adams for mechanical systems and EMTP for power systems. All of these programs provide extensive model libraries that can be used to simulate systems within their particular domain. The drawback of these special purpose tools is that the models are closed, they can not be altered or even inspected by the user. Since they only contain models from one domain it is also difficult or impossible to use them for complex, heterogenous systems.

Ideally, there should be good model libraries available that are both open and extensible. By building model libraries for special domains in a general modelling language you provide both the domain knowledge that special purpose software has and the extensibility and possibilities for multi-domain modelling that general modelling software has. This is the goal of the Modelica effort, see Elmqvist *et al.* (1999). Modelica™ is an open standard for a general modelling language. The specification of the language is freely available and there are also free base libraries with models in different domains, see Modelica Design Group (2000). The goal of this thesis is to expand the Modelica effort into the thermo-hydraulic domain.

1.2 Contributions of the thesis

The main contributions of the thesis are:

- some principles for developing model libraries in equation-based modelling languages. These principles are discussed in Chapter 3 and illustrated in Paper II.

- the development of the two model libraries **K2** and ThermoFlow for thermo-hydraulic applications.

- the application of **K2** to the modelling of a thermal power plant. This is shortly presented in Paper I. More details are available in Eborn (1998b); Eborn and Nilsson (1996).

- demonstration of how ThermoFlow model components are used in an application with evaporation in a pipe.

The thesis contains three articles and one conference paper. Below, the content of the papers is briefly summarized. References to related publications are also given. All of the papers have been written with co-workers, so some notes on who did what are included.

Paper I

Nilsson, B. and J. Eborn (1998): "Object-oriented modelling of thermal power plants." *Mathematical and Computer Modelling of Dynamical Systems*, **4:3**, pp. 207–218. ©Swets & Zeitlinger, Netherlands. Used with permission.

Contributions

The paper presents the modelling principles behind the **K2** model library. There is also a section on a **K2** application, simulation of a thermal power plant.

A shorter version of this article was presented by Bernt Nilsson at the EuroSim conference in Vienna, Nilsson and Eborn (1995). Bernt Nilsson did a lot of the rewriting into an article, but most of the work on the library and the power plant application was done by me.

Paper II

Eborn, J., H. Tummescheit and F.J. Wagner (2000): "Development of a Modelica base library for modeling of thermo-hydraulic systems." To be submitted for journal publication.

Contributions

The paper describes ThermoFlow, a Modelica base library for thermo-hydraulic systems. The basic equations for the central classes of the library, the control volume model, are given, as well as short descriptions of some component model and examples of how the library can be used.

The development of the base library was done in close collaboration with Hubertus Tummescheit and Falko Jens Wagner, although I and Hubertus started the work already in 1998, before Falko joined us in 1999. The first version of the article is also the fruit of joint work, the basis for the article is the paper (presented twice) Eborn *et al.* (2000) and Tummescheit *et al.* (2000). I have extended the paper for journal publication.

Paper III

Eborn, J., H. Tummescheit and K.J. Åström (2000): "Flow instabilities in boiling two phase flow." To be submitted for journal publication.

Contributions

The paper gives a physical analysis for pressure drop of an evaporating liquid flowing in a pipe. The analysis leads to a low-order model that can describe pressure-drop oscillations. In simulations, the simple model is compared to more complex models, based on ThermoFlow components.

The work on this paper was done together with Karl Johan Åström. He had the idea for the analysis and some calculations. I finished the analysis and have done the simulations. Hubertus contributed with some good ideas for the mixed model presented in the article. A shorter version of the article has also been published in Eborn and Åström (2000).

Paper IV

Eborn, J. and J. Sørlie (1997): "Parameter optimization of a non-linear boiler model." In Sydow, Ed., *15th IMACS World Congress*, vol. 5, pp. 725–730. W&T Verlag, Berlin, Germany.

Contributions

This case-study shows an application of the modeling tool OMOLA and the parameter optimization tool IDKIT. The tools are used to do structural validation of the Bell-Åström drum-boiler model, which is also described in Chapter 4.

The case-study presented in this paper was done in close cooperation with James Sørlie. It uses a model definition interface developed during the same time, Sørlie (1997). The writing of the paper was done jointly, but I presented it at the IMACS congress in Berlin. Some further results of the project are given in Sørlie and Eborn (1997); Sørlie and Eborn (1998). There is also a brief discussion in Chapter 5.

2

General Aspects on Modelling

Abstract

In this chapter the fundamental concepts of physical modelling are introduced and some examples are given to demonstrate the differences between formulations. Comparisons of object-oriented modelling with other methods like block-diagram approaches are also made.

2.1 Introduction

This thesis deals with object-oriented modelling of physical systems. I was introduced to this field of research through my Master's thesis which was concerned with modelling of an industrial pneumatic control valve, see Eborn (1994); Eborn and Olsson (1995). In this chapter some of the general aspects on using different modelling paradigms will be given, since this is not covered in any of the published papers. The question addressed here is, what are the fundamental differences between traditional modelling approaches and equation-based modelling?

2.2 Modelling paradigms

Traditionally in control and in computer simulation, modelling has been made in a procedural, block-oriented manner. This tradition comes more from concern with computational aspects than from user concerns. The modeller has to perform the tedious work of transforming a physical description in terms of balance- and constitutive equations into an explicit ordinary differential equation system (ODE)

$$\dot{x} = f(t, x) \tag{2.1}$$

This form of mathematical model is almost a computational program. It is in fact a *procedure* for calculating derivatives of the states. There is a lot of well-proven numerical software that can be used to solve these differential equations. Many powerful commercial simulation packages exist which use this type of models, with libraries of predefined computational blocks. What these programs offer is to provide an interface to the numerical algorithms and a graphical way of programming a computational model, but they do not give any help to construct the model.

Object-oriented modelling on the other hand tries to describe each part of a system as an object with a certain behaviour. This modelling paradigm is equation- or constraint-based[1]. Each object is described by fundamental physical relations, laws of nature, and not by a procedural function relating inputs to outputs. With this way of modelling the user is more concerned with the interface to the model objects and the model equations than with the computational order. This paradigm relies on the symbolic methods that modern computing offers since the model equations before simulation need to be manipulated into a differential and algebraic equation system (DAE)

$$g(t, x, \dot{x}, v) = 0 \tag{2.2}$$

The difference is that this manipulation is done by the computer and not by the user. For a description of some of the symbolic methods used, see Mattsson (1995). The difference between the procedural and constraint formulations can be illustrated as in Figure 2.1. With the procedural formulation the user gives a function for the direction to move in the state-space. Constraint modelling on the other hand just gives a relation that defines *possible* behaviours for the system.

EXAMPLE 1
As an example consider the modelling of a pneumatic spring-return actuator, common in the process industry. It consists of a pneumatic chamber with a diaphragm connected to a spring. The diaphragm position, x, is controlled with the mass flow, $\dot{m}$, entering the chamber. A drawing and schematic of the system together with the constitutive relations are given

[1] This is in agreement with the terminology in computer science, where *procedural* and *constraint* programming are classes of programming languages, see e. g., Abelson and Sussman (1985). On pages 285–294 there is a nice example of a system for constraint propagation.

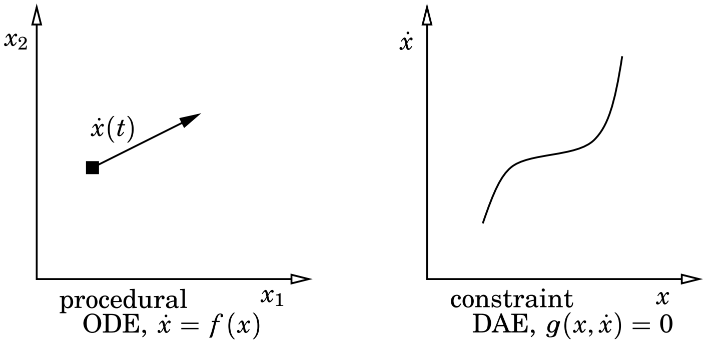


Figure 2.1 Different mathematical formulations of a model. An equation-based model defines constraints for possible behaviours in the state-space and not an explicit time-trajectory like a procedural model.

below. The ideal-gas law as it is used here assumes isothermal operation.

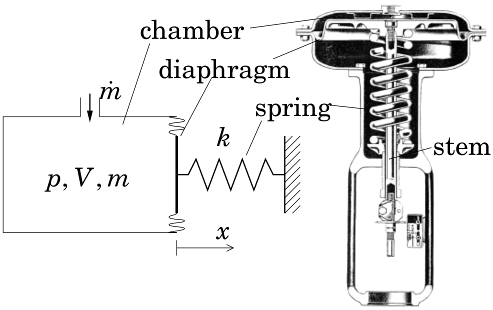
Constitutive relations:

Chamber :

$$\begin{aligned} \frac{dm}{dt} &= \dot{m} \\ V &= Ax + V_0 \\ pV &= m \cdot const \\ F &= pA \end{aligned}$$

Spring :

$$F = kx$$



To simulate this system in some block-diagram software, a possible representation could look like the diagram in Figure 2.2. Compare this with the object diagram in Figure 2.3. In the object diagram the connections imply relations between terminal variables, not computational causality as in the block diagram. Some of the physical structure of the system is lost in the block diagram. Also note that since the force, F, from the diaphragm depends directly on the position, x, there is an algebraic loop, i. e., a nonlinear algebraic equation system in the variables $\{V, p, F, x\}$. This loop is inherent in the system description, but often not handled very well by block-diagram software, although there are some special constructs to explicitly deal with algebraic loops, see Figure 2.8.

A possible extension of this model would be to account for the mass of

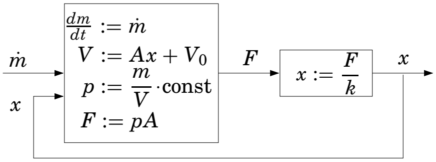

$\frac{dm}{dt} := \dot{m}$
$V := Ax + V_0$
$p := \frac{m}{V} \cdot \text{const}$
$F := pA$
$\dot{m}$
x
F
$x := \frac{F}{k}$
x

Figure 2.2 Block diagram of the pneumatic diaphragm.

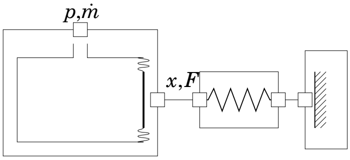


Figure 2.3 Object diagram of the pneumatic diaphragm.

the diaphragm and the attached stem. A force balance gives

$$m_s \frac{d^2x}{dt^2} = F_{\text{chamber}} - F_{\text{spring}}$$

This changes the block diagram into the one showed in Figure 2.4. Note that the description of the spring changes from $x := \frac{F}{k}$ to $F := kx$. The causality of the spring equation is reversed. This means that a model library with components applicable to this example would need to contain two *different* spring models, depending on the computational causality. Including mass in an object-oriented model would simply mean adding a mass object between the chamber and the spring in Figure 2.3. This would not in any way affect the descriptions of the spring or the chamber. □

A key point in the previous example concerns *causality*. This is a tricky subject which can be treated philosophically, as done in *Critique of Pure Reason* by Kant (1781), or more pragmatically. A thorough description of causality in the context of physical modelling is found in Strömberg (1994).

The concept of causality used here is *computational causality*, which gives the order of calculations to compute unknown variables from known ones. The point of Example 1 is that causality is a property of the *system*, depending on the choice of inputs for a particular experiment. The spring relation $F = kx$ has no causality in itself. The computational causality is

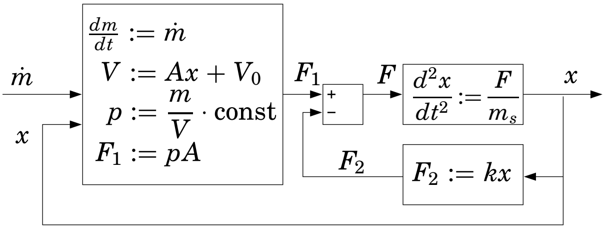

$\frac{dm}{dt} := \dot{m}$
$V := Ax + V_0$
$p := \frac{m}{V} \cdot \text{const}$
$F_1 := pA$
$\dot{m}$
x
F_1
+
-
F
$\frac{d^2x}{dt^2} := \frac{F}{m_s}$
x
F_2
$F_2 := kx$

Figure 2.4 Changed block diagram of the pneumatic diaphragm, mass included.

imposed on the spring by the choice of input ($\dot{m}$) and how the rest of the system is modelled (including mass or not). The fundamental drawback of procedural modelling is that it forces a causal description of every component of a system. Component models can then only be reused in exactly the same situation in the model of another system or experiment. *Causal* descriptions thus prevent *modularity*.

Block diagram modelling

Procedural models is the formulation used in almost all commercial block-diagram software. Building models of systems using input-output blocks has been the most common method in many engineering domains, especially in automatic control where the in-out formulation comes naturally from seeing a system as having manipulated inputs that affect the measured outputs.

Also for thermal power plants block-diagram models have been used a lot. The book Ordys *et al.* (1994) describes modelling and simulation of a general structure thermal power plant. The approach taken there is state-space modelling; breaking down the system in modules and for each of these modules determining inputs, outputs and states. This is possible through detailed analysis of each subsystem together with a global analysis of information flows. They also make some simplifying assumptions that decouples the modules, e. g., assuming constant pressure-drop across a valve which decouples it from the pressure downstream. This kind of assumption might be appropriate in a specific application, but it makes the model of the valve *application specific* and thus not reusable in a model

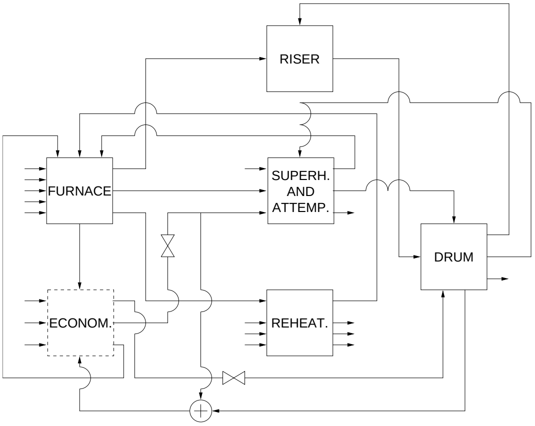


Figure 2.5 Block diagram of a boiler configuration, adopted from Ordys *et al.* (1994).

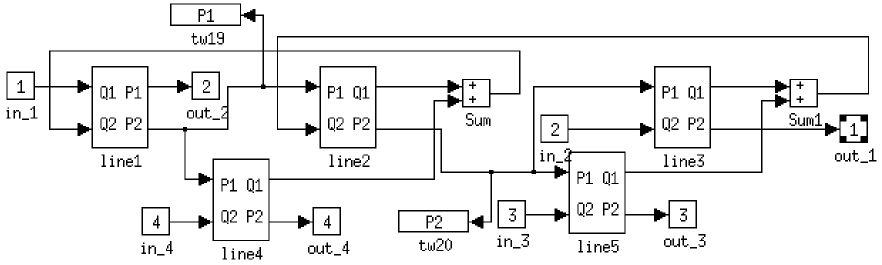


Figure 2.6 Block-diagram of upper part of the flow network in Example 2.

of another system.

Block diagrams for large systems tend to become very complex. As an example of this the block diagram of the boiler module in the Skegton unit described in Ordys *et al.* (1994) is shown in Figure 2.5. There are a lot of interconnections and dependencies between the blocks in the module. Each of these blocks are in turn described by block-diagrams or Fortran code.

EXAMPLE 2

As an illustration of the disadvantages of block-diagram modelling we look at a model for a cooling system, basically a network of tubes in which a hydraulic liquid flows. A block-diagram model of the marked part of the flow network in Figure 2.7 is shown in Figure 2.6. Note that the block diagram is considerably more elaborate than the system it represents, The problem with a procedural model of a flow network is that you need to pass information both in the forward and backward direction, giving rise

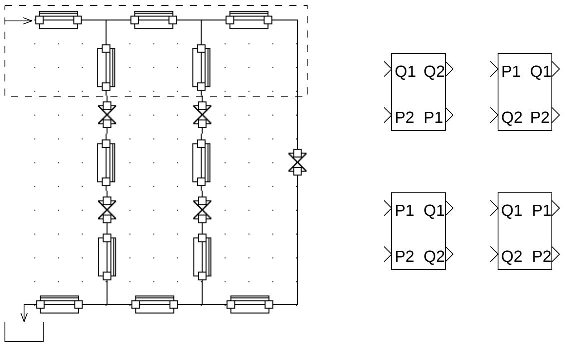


Figure 2.7 Flow network from Example 2 and four variants of a line model.

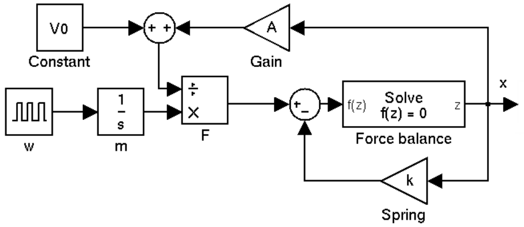


Figure 2.8 Simulink model of pneumatic actuator using constraint block.

to the complicated feedback connections between the line models. Another drawback is that the procedural formulation requires four variants of basically the same line model, depending on which of the two pressures, P, and two flows, Q, should be computed from the others. Two of the four variants are used in Figure 2.6, other variants in the rest of the network model. The need for several model variants is a major drawback, since changes to the line model means individually updating each of the variants. Manually changing several variants like that is a very error-prone procedure. □

One approach that can help the user working with causal modelling is to provide sorting of the equations. One of the first implementations of this was SIMNON, developed by Elmqvist (1972) at the Department of Automatic Control in Lund. This approach still requires that causal equations are specified, so you do need different models of a spring like in Example 1, but it can provide help with the causality between the blocks. A modern example of a software that uses this technique is EASY5® They claim, with some justification, that this is a major advantage over their competitors like Simulink® and SystemBuild®

It is true that sorting of equations makes it easier to build reusable model blocks, but still you cannot have a true model library unless you allow constraint models. In Simulink (from version 2.0) there is a special block called *Algebraic Constraint*. Using this you can build a constraint model of the system in Example 1, see Figure 2.8. The block called *Force balance* represents the constraint that the two forces from the chamber and the spring should balance. This is closer to equation-based modelling, since you can now always use the same form of the spring model, $F := kx$. If the mass of the stem should be included, the constraint block is replaced by the transfer function, $G_{xF}(s) = 1/(ms^2)$. This block thus allows constraint modelling in simple examples, but it is still far from being as

EASY5 is a trademark of The Boeing Company, Simulink is a trademark of The MathWorks, Inc. and SystemBuild is a trademark of Integrated Systems, Inc.

powerful as true equation-based modelling. It would be very tedious (and possibly inefficient) to try to model each flow in Example 2 with an algebraic constraint block.

This way of including constraint blocks with an explicit iteration variable is very reminiscent of the *tearing* technique used in Dymola.® With this technique the user chooses a suitable variable, called tearing variable, to break up an algebraic loop. The method is described in Elmqvist and Otter (1994). In Dymola version 4.0 automatic tearing is used, see Mattsson *et al.* (1999), which means that the tool automatically chooses an appropriate tearing variable, which simplifies modelling for the user.

Bond-graph modelling

An older but interesting paradigm that has its basis in physical analogies between different energy domains, like electrical and mechanical, was introduced by Paynter (1961). It is called *bond-graph* modelling and the name comes from that it is a graphical description of systems using bonds between elements. Each bond describes the power flow, which is the product of two conjugate variables, e. g., current and voltage in the electrical domain and velocity and force in the mechanical domain. Physical analogies show that capacitors and compliances (springs) can be described by a common C-element storing flow (current/velocity), and that inductors and inertias (masses) can be described by an I-element storing effort (voltage/force). An excellent book on modelling which gives a well-balanced description of bond graphs is Cellier (1991).

Bond graphs are very useful for simpler systems in the domains where there are natural power variables since graphical analysis methods exist that provide a lot of information of the system. For example, there are methods to automatically derive the computational causality of a bond graph and transform it to ODE form. However, bond graphs do not work that well in all domains, for example in thermodynamics you need to describe the energy interaction in up to three layers, creating a multilayer bond graph, since you must simultaneously consider mass, energy and momentum flows.

2.3 Object-oriented modelling

The modelling paradigm considered in this thesis is object-oriented modelling. It may also be called constraint modelling, non-causal or truly equation-based modelling as opposed to the procedural, block-oriented manner described previously. This is a slight confusion of two different

Dymola is a trademark of Dynasim AB.

concepts, since object-oriented refers to the structuring of the models, whereas constraint refers to the underlying description of the behaviour of the models. Nevertheless, both names are used in the thesis. Partly to emphasize the two different aspects, but also since the modelling languages that support constraint modelling usually also are object-oriented.

Object-oriented is a word that is sometimes misused, either for graphical tools referring to model blocks as objects, or because the software is written in some object-oriented programming language like C++. The "true" meaning of object-oriented modelling should be that the modelling language has some of the properties that object-oriented programming and design has, see Rumbaugh (1991); Booch (1991). Properties like the *class* concept, *inheritance, abstraction* and *specialization*. In object-oriented modelling each model is treated as an object, described by a class, which can be seen as a blue-print of the model. The class has attributes which can be locally defined or inherited from a *superclass*. Attributes can be either simple variables and equations or other objects. Through inheritance a common structure of a group of objects can be defined in a superclass, while different internal descriptions are kept in the subclasses. The subclasses are said to be specializations of the superclass.

Abstraction is a powerful tool to support complex system modelling. It implies the possibility to use a model without detailed knowledge of its internal structure or description. Necessary information to use a model object should be kept in its interface, which includes parameters and terminal variables. The interface contains all parts of the model that can be accessed from the outside. The internal behaviour description can be hidden, or *encapsulated*, and should not be accessed by other objects.

A more detailed explanation of the concepts of object-oriented modelling and of the modelling language OMOLA can be found in Andersson (1994). A description of the modelling language Modelica™ can be found in Elmqvist *et al.* (1999). Modelica is an open language specification given by the Modelica Design Group (2000).

Some of the advantages of using constraint modelling were mentioned in Section 2.2. There are also many advantages of using object-oriented structuring, these are more discussed in Section 3.3. However, there are also some drawbacks that should be mentioned. While constraint modelling relieves the user of doing manual manipulations of equations, this also means putting requirements on the simulation software used. The software must be able to perform the symbolic manipulations necessary to find the computational causality of the system equations. Sometimes algebraic constraints between dynamic equations gives a numerically more difficult problem, with higher DAE index. The software must then also be able to reduce the DAE index, see Mattsson *et al.* (2000). In some cases it may also be more numerically efficient to write the equations on explicit state space form, for example in combination with external functions for medium properties. External functions have a given causality. It may then be more efficient to have the inputs to the functions as states, since this avoids numerical iterations.

3

Building Model Libraries

Abstract

In this chapter some guidelines on how to build model libraries are given, from structuring guidelines to examples of model components. To give a completely general view on this subject is very difficult and possibly also harder to read and understand. The approach here is to draw some general conclusions from examples and from the experience gained during the development of the model libraries **K2** and ThermoFlow.

3.1 Developing new model libraries

There are a great deal of things to consider during the development of a new model library. Some of the more important issues are explained in the following sections. Among these are

- Purpose of the library
- Structure of models
- Compatibility with other models
- Numerical efficiency
- Examples and test models

Although this thesis can not claim to give a complete treatment on how to develop new model libraries, the most important issues will be covered. The points made here are general to equation-based and object-oriented modelling tools, but they are exemplified with the development of the model libraries **K2** and ThermoFlow. These were developed using two different modelling languages, OMOLA, see Andersson (1994), and Modelica, see Modelica Design Group (2000). Thus the chapter also contains some comments on differences between the modelling languages.

3.2 Purpose and goals

The first decision to be made in any modelling activity is to decide on the purpose of the model. This is also true for the design of a model library. What should be covered and what should be excluded from the library? Who are the intended users? These decisions affect the design from the very beginning. Expanding the scope of the model library can be very difficult, e. g., once the model interfaces have been specified.

The model library **K2** was from the beginning limited in scope to thermal power plants. The purpose of this library was to be able to model and simulate the control of a specific power plant, described in Paper I. This lead to the decision not to include momentum dynamics, since the frequency of those dynamics are outside the frequency range of a power plant controller. Flows in the plant were also restricted to one direction, since only normal operation of the plant was to be simulated.

ThermoFlow on the other hand has been built with the purpose to be applicable to general process applications where the thermo-hydraulic behaviour is the main concern. It is also intended to be extensible to applications that were not specifically in mind during the design. This means that the library has to be built in a flexible way and accommodate many different choices. For example to include different kinds of interfaces, for static or dynamic flows of either single- or multi-component fluids. The interfaces must also be able to handle reversing flows, since this is important in many applications, e. g., on-off control of refrigeration systems or start-up and shut-down procedures in industrial processes.

3.3 Structuring and decomposition

Object-oriented structuring is a powerful tool, that should be used with caution. The structuring should be based on the decomposition of a system into individual objects, and then the further decomposition of the objects themselves. A thorough discussion of object-oriented structuring, reuse and decomposition is given in Nilsson (1993). The treatment here is much shorter, but also explains the use of some of the concepts that were only suggested in the previous work of Bernt Nilsson.

The term decomposition is often used for breaking a system into parts. However, there are two different kinds of decomposition. The most com-

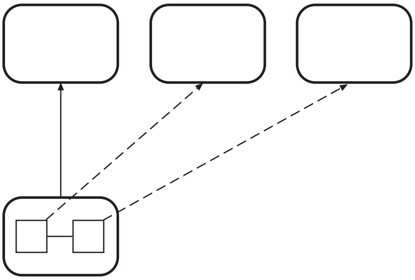


Figure 3.1 Object decomposition. The decomposed object owns (dashed arrows) two other objects.

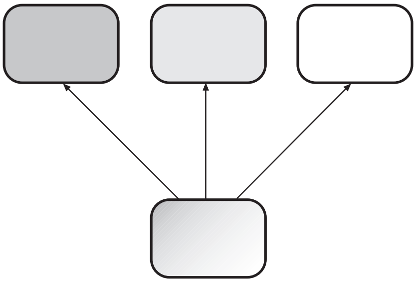


Figure 3.2 Subject decomposition. The decomposed object inherits (full arrows) from three partial classes.

mon one, structural[1], or *object* decomposition, shown in Figure 3.1, is not exclusive to object-oriented modelling. Components in any graphical modelling tool, e. g., Simulink, can be referred to as objects[2]. The other kind, *subject* decomposition, deals with the inheritance structure, see Figure 3.2. The arrows in Figures 3.1–3.2 show which class an object is derived *from*. The arrow points at a superclass which is inherited (full arrow) or is part of an aggregation (dashed arrow). The different kinds of decomposition are further explained below.

Object decomposition

Object decomposition is the subdivision of a system into its parts, e. g., a power plant into boiler, turbine and condenser, or in a more abstract way, the splitting of a heat exchanger into the machine and the liquid medium flowing in it, so-called *medium-machine* decomposition, see Nilsson (1993). Object decomposition is the more concrete way of sub-dividing systems and objects and should preferably be used for structuring of libraries with ready-to-use component models. In ThermoFlow for example, this subdivision is used in the sub-library Components.

Subject decomposition

In object-oriented languages, common properties of a group of objects are collected in a superclass, which the objects can inherit from and specialize. If the language has possibilities for multiple inheritance, as for example in Modelica, the common properties can be decomposed into different

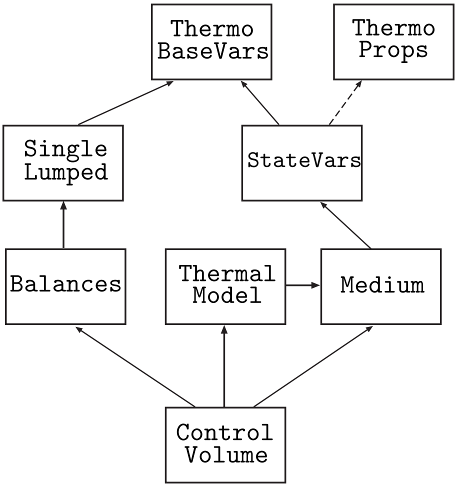
[1] Structural decomposition is the term used in Nilsson (1993), referring to the component structure of a model.

[2] Sometimes graphical component-based modelling tools are due to this called object-based, but the use of the term object is here limited to true object-oriented languages, as defined in Chapter 2.

partial classes, as illustrated in Example 1. In Nilsson (1993) multiple inheritance was proposed as an alternative to solving the *medium-machine* decomposition problem mentioned above. A possible problem with multiple inheritance stated there is that the partial classes need to refer to variables in each other. In ThermoFlow this is solved by having a common *variable set*, which all the partial classes inherit and use. Thus the names of all variables are explicit in the partial classes, and there is no problem with any naming convention. This is illustrated by an example.

EXAMPLE 1

Subject decomposition of a control volume model in ThermoFlow is illustrated in the figure. `ControlVolume` is split up into the `Balance` model containing the mass and energy balance equations, the `ThermalModel` with differential equations for state variables and the `Medium` model, that holds all property calculations. All these three parts inherit different versions of the variable set `ThermoBaseVars` which holds all the basic thermodynamic variables. The actual choice of state variables is made in the `StateVars` model, since the medium model and thermal model depend on this choice.



Full arrows indicate inheritance, the dashed arrow indicates that `StateVars` has a `ThermoProps` attribute.

□

The subject decomposition should be made in such a way that the partial classes are in some sense “orthogonal”, or independent. The partial classes `ThermalModel` and `Medium` in the previous example both depend on which states are used, e. g., $\{p, h\}$ or $\{\rho, T\}$. However, `ThermalModel` does not depend directly on what liquid the medium describes, thus the medium model is *replaceable*. In the same way `ThermalModel` only needs the sums of the inflows from `Balances`, but it does not care if these sums come from one, two or more inflows, which depends on what balance model is used.

The distinction between object and subject decomposition is based on the method used for composition, aggregation or multiple inheritance. Booch (1991) gives a spectrum of reasons for abstraction, where the first

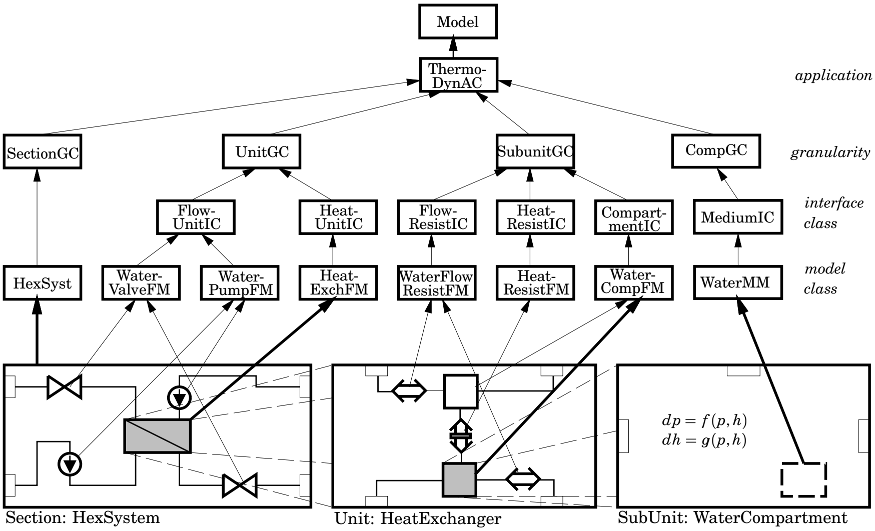


Figure 3.3 Class tree showing the inheritance structure of the **K2** library.

two are *entity* and *action* abstraction. This resembles the distinction made here, since object decomposition is based on physical entities, while subject decomposition focuses on the action/function of the partial classes.

Structuring

Like decomposition, structure is also a term with many meanings; library structure, component structure and inheritance structure for example. Structuring of a library and its models should serve several purposes, firstly to make the models in the library easy to find and easy to use, secondly, to increase the reuse of code and thus the maintainability of the models in the library. In an extreme idealized case each individual equation would only appear in one place in the library. Necessary changes then only have to be made in one place. Such extreme cases may not be practical, but clever use of structuring significantly decreases the number of occurrences of a single equation.

Structuring of K2

Because of the differences between OMOLA and Modelica the structuring of the libraries **K2** and ThermoFlow is different. The structure of the **K2** library is explained in Eborn and Nilsson (1996); Eborn (1998b). It follows the structuring guidelines given in Nilsson (1993) which recommends the hierarchical levels *plant – plant section – unit – subunit* for the object

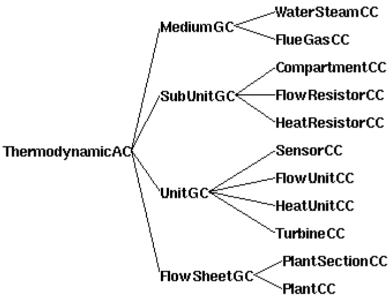


Figure 3.4 Upper levels in the **K2** class tree. Leaves in the tree correspond to sub-libraries.

decomposition. An example system is shown in Figure 3.3. Since single inheritance is used in OMOLA the inheritance structure builds up a class tree which is also shown in Figure 3.3. The inheritance structure was also used to divide the **K2** library into sub-libraries, which in the simulation environment OMSIM all appeared in a flat structure. The upper three levels of the **K2** class tree is shown in Figure 3.4, the lowest level corresponds to the sub-libraries in **K2**.

The availability of the concepts of packages and multiple inheritance implies that library structuring in Modelica can be radically different from OMOLA. Since Modelica packages (corresponding to sub-libraries) can be nested, they can also have a structure. The structure is used to make the library more accessible and to hide some of the *base class* packages, since these contain partial classes that the normal user is not interested in. Multiple inheritance makes it possible to use subject decomposition. The structure from this decomposition has been used in ThermoFlow for the structuring of the package `BaseClasses`. Multiple inheritance also has a drawback. The class tree becomes difficult to visualize and understand since it will become a rather complicated network.

Structuring of ThermoFlow

Structuring of the packages in ThermoFlow is done according to the two principles discussed previously. The user part of the library, the package `Components`, is structured using object decomposition. The user thus easily finds unit models of different applications in the library. The overall structure of ThermoFlow is shown in Figure II.4 on page 86. The basic part of the library, the `BaseClasses` package, is structured according to the subject decomposition of a control volume, as shown in Example 1. The thermodynamic control volume is the single most important model of the ThermoFlow library. The equations are given in Paper II. The package structure of BaseClasses is shown in Figure 3.5. The partial classes that build up a control volume model are in the packages Balances, StateTransformations and MediumModels. A complete model of a discretized control volume also holds a flow description, which is in the FlowModels package. The packages that concern the model interfaces are split up in four more sub-packages. This is because the interfaces are different depending on the choice of static or dynamic flow description and whether the medium is a single- or multi-component medium. Hence the package names SingleStatic etc.

3.4 Interfaces and compatibility

Interfaces, connectors, cuts, terminals, etc., are some of the names for the connecting elements between objects in different modelling languages. The choice of interfaces is, maybe, the most important structuring decision when building a model library, and definitely the most important when it comes to achieving *compatibility* with other model libraries. This is the reason for emphasizing standardized interfaces so much in the design of the Modelica base library. Design of interfaces is also important for the behaviours that can be handled by the models in a library, since there should be no other information exchange between objects in a model. This is discussed further in the next section.

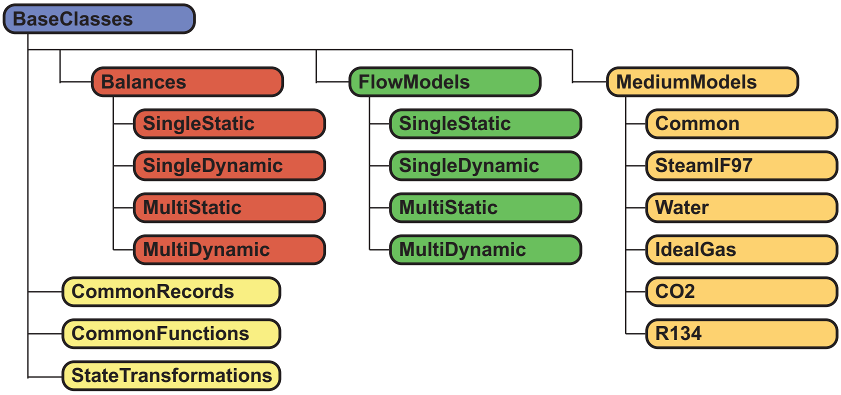


Figure 3.5 Package structure of base classes in the ThermoFlow library.

Reversing flows

Extra information is required in the connectors if the models are supposed to handle reversing flows. In principle all transported properties, e. g., enthalpy and composition, should be taken from the upstream direction. This can be handled in different ways. The simplest alternative, which possibly may be inefficient, is that the properties are included twice, upstream and downstream. The component can then choose which one to use depending on the flow direction. Another, elegant, solution that has been proposed is to have mutually exclusive if-then statements in different objects, see Ramos González (1994). Then you only need to have the transported properties once in the connector, but they are taken from different sources when the flow direction changes. This does, however, require a special implementation, as is shown in Example 2.

EXAMPLE 2

Model equations:

Volume 1:

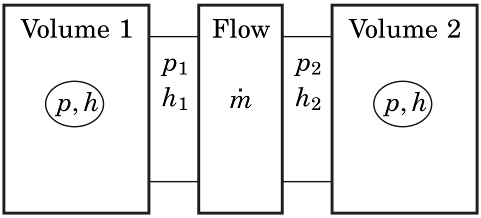
```
p1 = Vol1.p
h1 = if mdot>0 then Vol1.h;
dU = -mdot*h1;
```

Flow model:

```
mdot = A*sqrt(p1-p2);
h1 = h2;
```

Volume 2:

```
p2 = Vol2.p
h2 = if mdot<0 then Vol2.h;
dU = mdot*h2;
```



The example equations give `h2=h1=Vol1.h` when the mass flow is positive. This gives an energy flow from Volume 1 to Volume 2. Conversely, `h2=h1=Vol2.h` when the mass flow is negative. The simulation software must however realize that the two incomplete if-statements should be combined into

```
h1 = h2 = if mdot>0 then Vol1.h else Vol2.h;
```

which causes the need for a special implementation. □

Reversing flows in ThermoFlow

The **K2**-library did not handle reversing flows. In ThermoFlow it is handled by including extra flow information in the connectors that depend on the direction of the flow; convective heat flow q_c in single-component connectors and also component mass flows $\dot{m}_x$ in multi-component connectors. The information needed for the mass and energy balances is then contained in variables depending on the flow direction. The resulting equations from this choice are shown in Example 3.

EXAMPLE 3

Ideal three-ports have zero volume and thus represent stationary balance equations. In the fixed flow-direction case this is a very simple model. When any of the flows can change direction a tricky if-then structure is required to determine the enthalpy of the outflow. With the choice of interface variables used in ThermoFlow, the balance equations can be retained in their original form as zero-sum equations and the pressure and enthalpy in the three-port become algebraic states, determined by the simulation software. The equations are illustrated below.

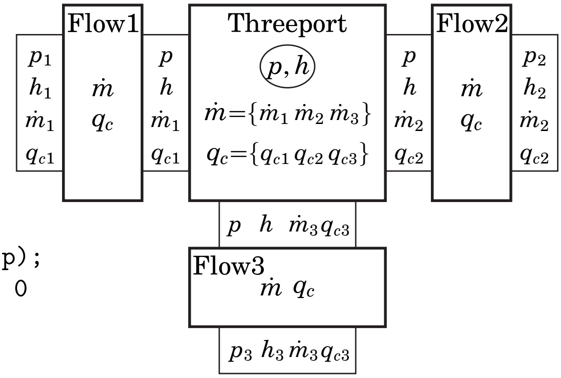
Model equations:

Threeport volume:

```
0 = sum(mdot);
0 = sum(q_conv);
```

Flow models:

```
mdot = A*sqrt(p_i-p);
q_conv = if mdot > 0
  then mdot*h_i
  else mdot*h;
```


Flow1
Threeport
Flow2
Flow3
p_1 h_1 $\dot{m}_1$ q_{c1}
$\dot{m}$ q_c
p h $\dot{m}_1$ q_{c1}
(p,h)
$\dot{m}=\{\dot{m}_1\,\dot{m}_2\,\dot{m}_3\}$
$q_c=\{q_{c1}\,q_{c2}\,q_{c3}\}$
p h $\dot{m}_2$ q_{c2}
$\dot{m}$ q_c
p_2 h_2 $\dot{m}_2$ q_{c2}
p h $\dot{m}_3 q_{c3}$
$\dot{m}$ q_c
p_3 $h_3 \dot{m}_3 q_{c3}$

The resulting equation for the enthalpy in the three-port is

$$h = \sum_{\{i:\dot{m}_i>0\}} q_{c,i} / \left(\sum_{\{i:\dot{m}_i>0\}} \dot{m}_i \right) \tag{3.1}$$

The equations for pressure or composition (in the multi-component flow case) are similar, but they are more involved and therefore not displayed here. The singular case, when $\sum \dot{m}_i = 0$, is not severe, since it only occurs when all flows are identically zero. If necessary, simulation software like

Dymola can use continuity to deal with indeterminate expressions. Using such software the limit of h in (3.1) will be calculated correctly even when both numerator and denominator are zero. □

The solution to have convective heat flow in a flow connector was mainly chosen to allow multiple flows in one node, i. e., using the *flow semantics* of Modelica. By connecting many flow models to one volume connector all the mass flows and convective heat flows will be summed via the zero-sum equation generated by the connection.

Considering only one-dimensional flow, flow semantics could be a feasible construction for the momentum in the dynamic flow case. The momentum is, however, a quantity associated with a direction and it should be represented by a vector variable. The goals of the ThermoFlow library is only to consider one-dimensional flow, but the practical use of one-dimensional momentum balances is limited to straight pipes. For example in a split of a flow into two branches, the angle between the branches should be 0° for the momentum to be preserved. Since this is not a reasonable model assumption, the momentum balance must be treated in special models which do not use the flow semantics of Modelica.

Heat flow interfaces

Heat flow interfaces seem rather straightforward, because there is a potential variable, temperature T, that drives the flow variable, heat flow rate q. This is true, but there is a problem because the heat flow resistance is split up in boundary layer and wall resistances. In **K2** all effects were lumped in resistance objects within a wall model. Since the calculation of the boundary layer resistance requires some properties from the adjoining volumes, extra information about the liquid medium must then be passed in the heat flow connection.

The inclusion of the boundary layer description in the wall model is not natural. In ThermoFlow the boundary layer can instead be included in the adjoining volume. This makes the wall model a pure wall, that can be modelled as static or dynamic by including thermal storage effects. The wall resistance can also be neglected by leaving out the wall model entirely. The decomposition of the heat flow equations into different objects also makes it possible to build many different model combinations, e. g., neglecting one or both of the boundary layer resistances or the wall resistance. This flexibility could lead to problems in certain cases, but it does not, as is shown in Example 4.

EXAMPLE 4

The trickiest case of heat flow modelling through a wall is when all heat flows are modelled with static models and both wall and boundary-layer resistances are included. The model introduces five unknowns, one temperature on each side of the wall and three heat flows, which should be solved for by three heat flow equations and the static condition that all flows are equal. A simple example of the structure of the equations is given below.

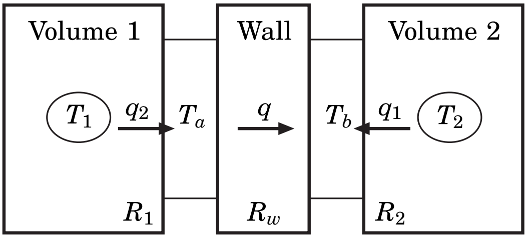
Model equations:

Wall resistance:

```
q = (Ta - Tb)/Rw;
q = q1;
q = -q2;
```

Boundary-layer resistances:

```
q1 = (T1 - Ta)/R1;
q2 = (T2 - Tb)/R2;
```



In this system the temperatures in the adjoining volumes, T_1 and T_2, are assumed to be given (states). The other five unknowns should be solved for. In this simple example the result is a system of linear equations in T_a and T_b. This system always has a solution for finite values of R_1 and R_2.

$$\begin{bmatrix} 1/R_1 & 1/R_2 \\ -1/R_w & 1/R_2 + 1/R_w \end{bmatrix} \begin{bmatrix} T_a \\ T_b \end{bmatrix} = \begin{bmatrix} T_1/R_1 + T_2/R_2 \\ T_2/R_2 \end{bmatrix}$$

In other cases, for example when the logarithmic mean temperature difference[3] is used, a system of nonlinear equations has to be solved.

An even simpler case is when the wall resistance, R_w is neglected. The wall model then reduces to the equation $T_a = T_b$ and an explicit solution is

$$T_a = T_b = \frac{T_1 R_2 + T_2 R_1}{R_1 + R_2}$$

□

One combination of heat flow models that leads to potential problems is when all resistances are neglected. This is a rather degenerate case, leaving the temperatures in the adjoining volumes equal. Constraining the temperatures in two volumes results in a DAE problem with index 2,

[3] The logarithmic mean temperature, ΔT_{lm}, is used to obtain a correct static behaviour in lumped models of heat transfer. It uses the static analytic solution of the heat transfer equation, which gives an exponential temperature profile.

much like the system with connected compartments discussed in Paper I. Even this degenerate case may in some situations be solved symbolically, for example with ideal gas models, when T is a state variable in the adjoining volumes.

3.5 Examples and test cases

The importance of good test examples should not be underestimated. Almost all design activities are iterative in nature, and running test examples is the "proof of the pudding" for a model library. Realistic examples are essential for thorough testing of a model library. Good tests will reveal inconsistencies and errors both in the library structure and in the actual models. It can, however, be a difficult task to find examples that test all aspects of a library. This is especially true for a model library like ThermoFlow which is supposed to be general in nature and useful in many different application domains.

The use of test examples in **K2** and ThermoFlow can be seen as two extremes on how to use test examples. With **K2** the goal from the beginning was to simulate a specific plant, the biogas thermal power plant in Värnamo, see Eborn and Nilsson (1996) and Paper I. This plant was used as a test example throughout the design. This meant that almost all components in the library was built for this purpose, although many of them could be reused in later projects to model other plants, see Klevhag (1996); Stojnic (1997); Löfgren and Svensson (1997); Eborn (1998).

On the other hand, the goal of ThermoFlow was from the beginning to build a base library applicable for many different thermo-hydraulic problems and domains. During the earlier stages of the library design, smaller examples like a model of pressure waves in a pipe, simple plate heat exchangers and a model of a combustion engine were used for testing the library. Later, more complex examples, like the boiler pipe model in Paper III and a fuel cell model provided ample testing. These tests of the design at different stages, often lead to redesign of some of the basic structures. Although the basic ideas of the models in ThermoFlow have remained the same throughout the design, the structure of the base classes has undergone three major reconstructions.

Feedback from good alpha-testers is also valuable. Parts of the library have been used for modelling of refrigeration systems, see Bauer (1999); Pfafferott and Schmitz (2000). An early version of the library was also used for a model of a paper dryer section, see Pontremoli (2000).

3.6 Concluding remarks

The two different approaches taken to build **K2** and ThermoFlow show very clearly how the goal of a library is reflected in the design. A simple, application-oriented library sacrifices structure and generality, while the design of a basic library with wide applicability needs a lot of work and constant reconsideration. You should not fall in love with your first design, but also be careful not to change too easily. It is easy too overlook consequences of changes and reverting to a previous version can be very tedious. Using a good version control system like CVS, at CVShome.org (2000), is very useful for a single developer and essential in a multi-developer project.

4

Modelling Examples

Abstract

Two examples that illustrate the advantages of equation-based modelling are discussed in this chapter. The examples are used to compare object-oriented modelling with traditional methods.

4.1 Drum-boiler model

In a thermal power plant cycle, water is first evaporated to steam, which is then super-heated, expanded through a turbine and then condensed back to water. The evaporation is done in a boiler, which can be of two different types; a drum-boiler or a once-through boiler. The difference is that the drum-boiler has a drum that separates the liquid water from the steam, while once-through boilers produce steam directly in continuous flow through a pipe. This section discusses a model of a drum-boiler. Section 4.2 has a short discussion on the boiler pipe model in Paper III, which could be used for simulation of a once-through boiler.

Modelling and control of thermal power plants is a classical control problem, treated in Chien *et al.* (1958); Eklund (1971); Lindahl (1976); Kwatny and Berg (1993). One of the key difficulties is the control of the water level in the drum-boiler. Water level dynamics are non-minimum phase, because of the so-called shrink and swell effect, and vary very much with the load. At low loads the non-minimum phase behaviour is much more pronounced, which severely limits the performance of the level control. This means that the control design must be either very conservative or use gain-scheduling to account for the varying dynamics. Design of the water level control for a nuclear power plant was posed as a benchmark problem in Bendotti and Falinower (1999). The successful solutions to

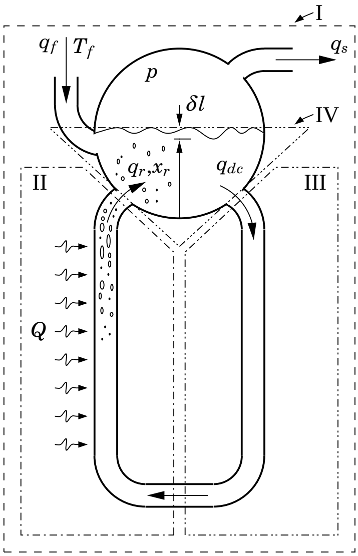


Figure 4.1 Schematic of a drum-boiler with main variables and control volumes.

the benchmark problem all used gain-scheduling, see Ward and Middleton (1999); Eborn *et al.* (1999a).

With the deregulation of the power market there are also increased requirements on the control performance of thermal power plants. The increased performance requirements are difficult to meet without more detailed knowledge of processes like the drum-boiler. There is a long tradition in Lund of developing low order nonlinear models of the drum-boiler, concluded in Bell and Åström (2000). The basic equations for this model are both simple and transparent, but many manual operations need to be made to reduce the model to a format that is convenient for simulations. By using Modelica and Dymola the model can be entered in its basic form and all manipulations are done automatically. It is thus a nice example of the power of this modelling technique.

Pressure dynamics, second-order model

A simple model of the drum-boiler, that captures the pressure dynamics very well is a second order model based on the global mass and energy balances. The same notation as in Bell and Åström (2000) is used here: V denotes volume, ρ density, h specific enthalpy, T temperature and q mass flow rate. Subscripts are s, w, f and m for steam, water, feedwater[1]

[1] Feedwater denotes properties of the inflow to the drum-boiler.

and metal, respectively. The subscript t is used to mark a total quantity, i. e., taken over the global system.

The global mass and energy balances is taken over the control volume marked I in Figure 4.1. This gives

$$\begin{aligned} \frac{dM_t}{dt} &= \frac{d}{dt}[\rho_s V_{st} + \rho_w V_{wt}] = q_f - q_s \\ \frac{dE_t}{dt} &= \frac{d}{dt}[\rho_s h_s V_{st} + \rho_w h_w V_{wt} - pV_t + m_t C_p T_m] = Q + q_f h_f - q_s h_s \end{aligned} \tag{4.1}$$

where Q is the heat flow rate into the system and C_p is the heat capacity of the metal. The term $-pV_t$ comes from replacing specific internal energy u with $h = u - p/\rho$. It is usually negligible compared to the other terms.

By also including the relation between the volumes

$$V_t = V_{st} + V_{wt} \tag{4.2}$$

(4.1) can be rewritten into a state equation in pressure and water volume, which is done in Bell and Åström (2000). Although the rewriting of equations is fairly straightforward for a simple model like this it soon becomes tedious work for larger models.

In Modelica the model can instead be given in its basic form, as a differential-algebraic equation system, and then the simulation software determines the state-space realization. This means that the second-order model is given as in (4.1–4.2) directly. The Modelica code for this model is given below.

```
model DrumBoiler2ndOrder "Basic balance equation model"
  extends BoilerShell(p(start=8.5e6,fixed=true),
    redeclare model SaturationMedium=SaturationMM);
  parameter SIunits.Volume Vt=88 "Total volume";
  parameter SIunits.Mass mt=300e3 "Total metal mass";
  parameter SIunits.SpecificHeatCapacity Cp=550 "Metal Cp";
  SIunits.Mass Mt(fixed=false,start=1e3) "Total mass";
  SIunits.Energy Et(fixed=false,start=1e6) "Total energy";
  SIunits.Volume Vst "Total steam volume";
  SIunits.Volume Vwt(start=57.5,fixed=true) "Total water vol";
  SIunits.Temperature T = pro.T + Modelica.Constants.T_zero;
equation
  Mt = pro.dv*Vst+pro.dl*Vwt;
  der(Mt) = a.mdot + b.mdot;          // Global mass balance
  Et = pro.dv*pro.hv*Vst+pro.dl*pro.hl*Vwt-p*Vt+mt*Cp*T;
  der(Et) = Q.q[1]+a.q_conv+b.q_conv; // Global energy balance
  Vt = Vst + Vwt;                     // Volume constraint
end DrumBoiler2ndOrder;
```

This model inherits from two other classes, `BoilerShell`, which specifies the mass and heat flow interfaces, and `SaturationMM`, which gives saturated medium properties as a record `pro = {T, dv, dl, hv, hl}`. The interfaces, `a` and `b`, contain mass flow rate `mdot` and convective heat flow, defined as `q_conv=mdot*h`$= qh$. The interface variables are the ones used in the ThermoFlow library and thus notation is not exactly the same as in (4.1). The reference direction for the variables in the interfaces are positive inflow, which means that `b.mdot` and `b.q_conv` usually are negative.

When Dymola is used to simulate this model, the software finds that it needs to differentiate the expressions for M_t and E_t to transform the system of equations into explicit form. This is possible if the medium properties are given as simple expressions, for example polynomials. It is also possible to supply the derivative function if the properties are given by external functions or non-differentiable expressions. With these derivatives Dymola generates explicit code with p and M_t as states.

Drum-level dynamics, fourth-order model

Although the model in the previous section describes the pressure dynamics very accurately; it is not very useful since the difficult control problem is the level control. The balance equations for the control volumes (CV) marked $II - IV$ in Figure 4.1 can be used to include an accurate description of the water-level dynamics. In Bell and Åström (2000) the level dynamics are included via a static momentum balance for CV III, mass and energy balances for CV II and a mass balance for steam in CV IV. The quantities in these control volumes are marked with subscripts dc for down-comer, r for riser and sd for steam under the drum water level. The balance equations can be written as

$$\begin{aligned}
0 &= (\rho_w - \rho_s)\alpha_v V_r g - \frac{k}{2}\frac{q_{dc}^2}{\rho_w A_{dc}} && (4.3)\\
\frac{dM_r}{dt} &= \frac{d}{dt}[\rho_s \alpha_v V_r + \rho_w(1-\alpha_v)V_r] = q_{dc} - q_r && (4.4)\\
\frac{dE_r}{dt} &= \frac{d}{dt}[\rho_s h_s \alpha_v V_r + \rho_w h_w (1-\alpha_v)V_r - pV_r + m_r C_p T]\\
&= Q + q_{dc}h_w - q_r(\alpha_r h_c + h_w) && (4.5)\\
\frac{dM_{sd}}{dt} &= \frac{d}{dt}[\rho_s V_{sd}] = q_r\alpha_r - q_{sd} - q_{cd} && (4.6)
\end{aligned}$$

By combining the mass and energy balance for the riser, this finally gives a fourth-order model. The combination of the two balance equations for CV II was done by hand manipulations which not only are tedious, but also error-prone.

By simply adding the balance equations (4.3–4.6) to the model in Section 4.1 a model with 5 dynamic balance equations is obtained. However, there are only 4 states, since the assumption that there is only one pressure in the system imposes a constraint on the balance equations. This way the same model as in Bell and Åström (2000) is obtained, but without the hand manipulations. The Modelica code for this model is given below, note that all the equations from the second-order model are also inherited into this model.

```
model DrumBoiler4thOrder
  // from Åström-Bell, Drum-boiler dynamics, Automatica.
  //   four-state, five diff-equation model, fully implicit
  extends DrumBoiler2ndOrder;
  parameter SIunits.Volume Vr=37 "Volume of risers";
  parameter SIunits.Volume Vdc=11 "Volume of downcomers";
  parameter SIunits.Mass mr=160e3 "Riser metal mass";
  parameter SIunits.Mass md=100e3 "Drum metal mass";
  parameter SIunits.Area Adc=0.355 "Downcomer flow area";
  parameter SIunits.Area Ad=20 "Drum wet area";
  parameter Real k=25 "friction coefficient";
  parameter Real beta=0.3 "empirical qsd coefficient";
  parameter Real Vsd0=6 "Bubble volume coefficient";
  parameter Real Tsd=5 "Bubble volume time constant";
  constant Real g=Modelica.Constants.g_n;
  SIunits.Mass Mr(fixed=false,start=1e3) "Riser mass";
  SIunits.Mass Msd(fixed=false,start=10) "Steam bubble mass";
  SIunits.Energy Er(fixed=false,start=1e6) "Energy in riser";
  SIunits.Volume Vwd "Drum water volume";
  SIunits.Volume Vsdb(start=5,fixed=true) "Bubble volume";
  SIunits.MassFlowRate qdc "Downcomer flow";
  SIunits.MassFlowRate qr  "Riser flow";
  SIunits.MassFlowRate qcd "Condensation flow";
  SIunits.MassFlowRate qsd "Steam bubble flow";
  SIunits.Length dl "Drum level";
  Real am "steam volume ratio";
  Real xr(start=0.051,fixed=true) "steam mass ratio";
  SIunits.SpecificEnthalpy hc "Condensation enthalpy";
equation
  Mr = pro.dv*am*Vr + pro.dl*(1 - am)*Vr;
  // Riser mass balance
  der(Mr) = qdc - qr;
  Er = pro.dv*pro.hv*am*Vr + pro.dl*pro.hl*(1 - am)*Vr
     - p*Vr+mr*Cp*T;
  // Riser energy balance
  der(Er) = Q.q[1] + qdc*pro.hl - (xr*hc + pro.hl)*qr;
  am = pro.dl/(pro.dl - pro.dv)*(1 - pro.dv/xr
      /(pro.dl - pro.dv)*ln(1+xr*(pro.dl - pro.dv)/pro.dv));
  hc = pro.hv - pro.hl;
  // Static momentum balance
  pro.dl*Adc*(pro.dl-pro.dv)*g*am*Vr = k*qdc^2/2;
  Msd = pro.dv*Vsdb;
  // Bubble mass balance
  der(Msd) = xr*qr - qsd - qcd;
  qcd = (pro.hl*a.mdot-a.q_conv)/hc - der(p)*((Vwd + Vsdb) -
    (Msd*pro.dhvdp+pro.dl*Vwd*pro.dhldp + md*Cp*pro.dTp))/hc;
  qsd = pro.dv*(Vsdb - Vsd0)/Tsd + xr*qdc + xr*beta*(qdc-qr);
  Vwd = Vwt - Vdc - (1-am)*Vr;
  dl  = (Vwd + Vsdb)/Ad;
end DrumBoiler4thOrder;
```

This model needs the medium property derivatives not only for the manipulations, but they are also used in the expression for the condensation enthalpy, qcd.

When this model is simulated in Dymola the symbolic manipulation reduces the five differential equations to four state equations in the variables $\{p, M_t, M_r, M_{sd}\}$.

4.2 Boiler-pipe model

The equations of the boiler-pipe model will not be repeated here, since they are all given in Paper III. The Modelica code of the models is also given in Appendix IIIa. However, the boiler-pipe model, and also the previous drum-boiler example, is a good example of how different levels of complexity can be used in models for different purposes.

The difference between the different boiler-pipe models in Paper III is that more dynamics are added to include more complex phenomena. The simple model, $\mathcal{M}_1$, is just a static model of the pressure drop, which means that the pressure drop must always follow the given function, $\Delta p = f(\dot{m})$. The one-flow model, $\mathcal{M}_2$, adds the discretized energy dynamics in the pipe, which introduces a lag in the density variations. This lag affects the pressure drop and gives limit cycles of different amplitude, see Figure III.9 on page 113. By also adding the discretized mass dynamics in the fully discretized model $\mathcal{M}_3$, another phenomenon is introduced, pressure waves. This phenomenon is not interesting for the purpose of the model, to study the pressure-drop oscillations. Thus it can be concluded that model $\mathcal{M}_3$ is too complex for the purpose of the study. This compromise between desired behaviour and complexity in the model is necessary to recognize. There is no single best model, a model is always built for a specific purpose and for every purpose there is a different model.

5

Model Validation

Abstract

This chapter gives some comments to the work on model validation done together with James Sørlie, see Paper IV. It also contains a short literature review of model validation methods, mainly for physical models.

5.1 Introduction

Models of systems are almost always built with the purpose to draw conclusions about the real system from simulations or from an analysis of the model. Consequently issues of model validity are very important. The model must accurately describe the essential characteristics of the system. When you use a physical model, drawn from first principles, this means both a need to have accurate knowledge of the system parameters as well as having a correct model structure, accounting for important physical phenomena in the real system.

Although model validation is a well investigated area within the field of system identification, it has often been dealt with in a heuristic fashion for physical models. An excellent example of classical validation of a physical model is presented in Leva *et al.* (1999). In that article, a model of a drum-boiler is validated and ’tuned’ by comparing experimental step-response data to simulations. There are several reasons for this trial-and-error way of doing validation. The main reason is that physical models usually are nonlinear. Since the mathematics and stochastics for nonlinear systems are not as well developed as for linear systems, the task of model validation for physical models is a difficult one. Also, parameters in a physical model are always, to some extent, uncertain. This means that there is always some element of parameter tuning involved in the validation.

5.2 Model structure validation via parameter optimization

Ideas for an approach to model structure validation is described in Paper IV. The task of finding accurate model parameters as well as validating the model structure is addressed by doing nonlinear parameter optimization of different model structures. The approach mainly consists of comparing different model structures, or hypotheses, on different sets of measured data. The choice of the best model structure is based on Akaike's Information Criterion (AIC). Further studies, done after the submission of Paper IV, has been presented in Sørlie and Eborn (1997); Sørlie and Eborn (1998). The continued case-study compared a fifth-order model structure, $\mathcal{M}_5$, to the third- and fourth-order models in Paper IV. In $\mathcal{M}_5$ the delayed steam flow from the risers is added as a new state, but this

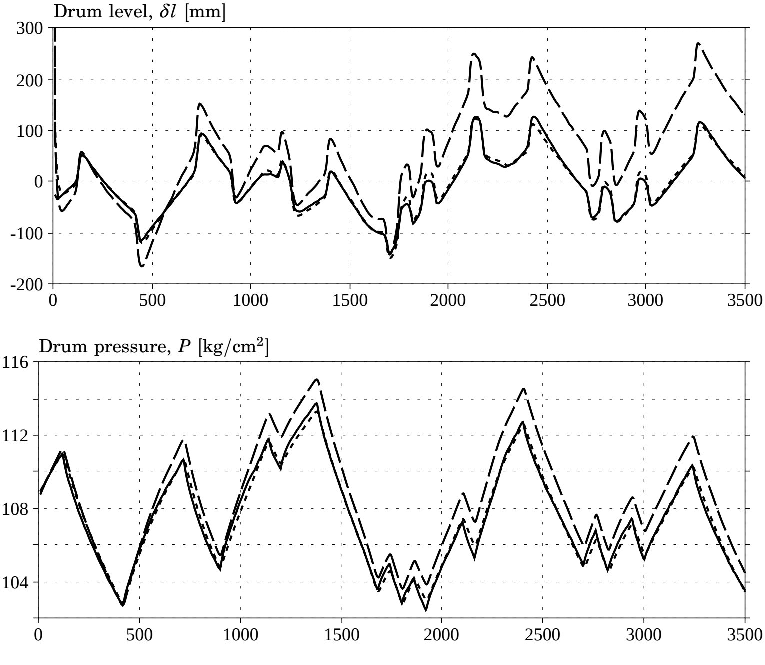


Figure 5.1 Comparison of measured data(—) to simulations of model with nominal(– –) and estimated parameters(- - -).

model was rejected since it did not give significantly better results than $\mathcal{M}_4$. However, with only minor changes in the $\mathcal{M}_4$ model structure the almost perfect fit[1] shown in Figure 5.1 was obtained. The changes mainly concerned initialization of the states. The coefficients of the state equations were also changed slightly due to the inclusion of internal energy instead of enthalpy. It is interesting to note that there are only small differences between the two curves with nominal and optimized parameter values in Figure 5.1. The main difference seen by the naked eye is the reduction of the drift in the simulated output. This improvement comes from optimization of the conversion factors from measured inputs. The good fit obtained for this and other data sets shows that the model structure $\mathcal{M}_4$ accurately describes the system. It can also be concluded that the model is useful for control design, since it captures all the fast dynamics even with nominal parameter values.

5.3 Model validation methods

The process of validating a model is sometimes called *external* validation, see Murray-Smith (1998). This is to stress the difference compared with (internal) verification, which only concerns the consistency of the numerical implementation of a mathematical model. Model validation seeks to find evidence of the level of agreement between the model and the real system. When working with model validation it is important to recognize that it is not possible to conclusively validate a model. According to Popper (1935) a theory can only be falsified, i. e., proven wrong, by experimental tests. By passing any number of tests, a model is only as yet unfalsified. The purpose of the model is also an important aspect that affects the model validation process. The accuracy requirements are often much higher on a model used for process design purposes, than on a model for control design. Recent activities within the research area *modelling and validation for control design* emphasize the accuracy in the frequency range important for control. This can for example be obtained by selection and prefiltering of the measurement data used for the validation, see Ljung and Guo (1997).

Model validation can be approached using a number of different methods, e. g.,

- Model validation in the context of system identification
- Robust model validation

[1] Note that there are three lines in the figure. The estimated parameters line (- - -) is hardly visible behind the measurements.

- The model distortion method, described in the next section.

Classical model validation methods are used together with system identification, usually using linear difference equation models, see Ljung (1987). The classical validation methods are based on statistical tests of the model residuals, i. e., the difference between the measurements and the predicted model output, see e. g., Ljung and Hjalmarsson (1995).

The robust model validation methods have emerged within the framework of robust control of uncertain systems, see Smith *et al.* (1997). There are a number of methods, either emphasizing frequency-domain properties or time-domain. They have the property in common that the model errors are not described by stochastic processes, but rather by worst-case bounds. By optimization procedures, minimal bounds on the model uncertainty and disturbances are obtained. The bounds are used for the rejection or acceptance of the model. The uncertainty bounds can also be used for robust control design. The models used within this framework are also principally linear, but there are also examples of validation of (simple) nonlinear models using robustness methods, e. g., Dullerud and Smith (1997).

The method of model distortion, on the other hand, directly addresses the problem of nonlinear model validation. It is also based on parameter optimization like the approach described in Paper IV. Therefore the model distortion method is described with some more detail in the next section.

5.4 Method of model distortion

The model distortion method is based on distorting model parameters to make the model output fit the measurements exactly, see Butterfield and Thomas (1986a); Butterfield and Thomas (1986b). The size of the distortions will then say something on the quality of the model; the better the model structure is, the smaller are the required distortions to fit the model to the data. It is not only based on parameter optimization, since the model distortions are time-varying.

The distortion method starts with parameter optimization that minimize some model error criterion, e. g., mean-square error. The *optimal* parameters, $\mathbf{b}$, give the nominal model

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{b}), \quad \mathbf{y} = \mathbf{C}\mathbf{x}$$

where $\mathbf{x}$ are the states, $\mathbf{u}$ the inputs and the outputs, $\mathbf{y}$, are a subset of the states. The parameters in the model are then distorted by time-varying parameter distortions, $\delta\mathbf{b}(t_j)$, to make the model output fit the measurements exactly,

$$\dot{\chi} = \mathbf{f}(\chi(t_j), \mathbf{u}(t_j), \mathbf{b} + \delta\mathbf{b}(t_j)), \quad \mathbf{z}(t_j) = \mathbf{C}\chi(t_j) \tag{5.1}$$

where $\mathbf{z}$ are the measurements and χ are the distorted states.

Butterfield (1990) gives two different criteria for the acceptance or rejection of a model, based on the acceptable variance, δb_k^2, for each model parameter, b_k. These acceptable variances are a-priori information, taken from known tolerances or approximations in the modelling. A time domain criterion is based on solving (5.1) for the distortions $\delta\mathbf{b}$ and comparing their variance with the acceptable variance.

$$\frac{\mathrm{Var}(\delta\mathbf{b}_k)}{\delta b_k^2} \leq 1$$

If this criterion is met for all parameters b_k then the model is accepted. Explicit expressions for the model distortions for linear discrete-time models are given in Cameron (1992); Cameron *et al.* (1998).

The second criterion is based on a simplified frequency-domain analysis. In this criterion the results from the nominal parameter optimization are used instead of the explicit model distortions. The shape of the mean-square error as a function of each model parameter, $J(\beta_k)$, is taken as a measure of the required model distortions. The principal shape of the error function is illustrated in Figure 5.2. It can often be locally approximated by a quadratic form. The chord length, Δb_k, at $J = 2J^*(b_k)$ is taken as a measure of the model distortions required to fit the model to the measurements. The frequency-domain criterion then sums the influence of all the model parameters,

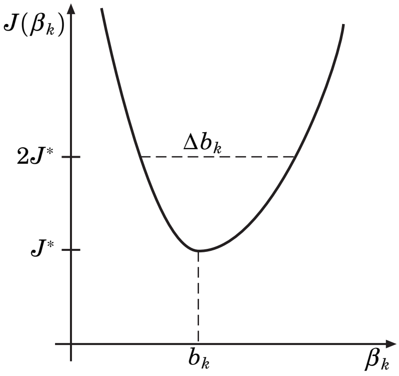


Figure 5.2 Mean-square error, J, as a function of the parameter value, β.

$$\sum_{k=1}^{p} \frac{\delta b_k^2}{\Delta b_k^2} \geq 1$$

where p is the number of parameters. This criterion is claimed to be somewhat pessimistic and the chord length can be shortened by a factor α. This factor is given by Butterfield and Thomas (1986a) as approximately

0.8, while Cameron *et al.* (1998) give a statistical derivation based on the F-distribution to calculate α for different significance levels as a function of the number of parameters and available data points.

The use of parameter optimization in the model distortion method resembles the way it was used in Paper IV. However, the distortion method gives a quantitative measure on model quality for physical models. It also takes into account a-priori information. The purpose of the model is given by the data used for the validation, which should reflect the kind of behaviour that the model should be able to reproduce. There are of course also draw-backs, for example that the method requires a model on explicit ODE form and that the analysis assumes that the measured outputs are also states in the model. These are not severe and could possibly be relaxed.

A model validation method based on parameter optimization should preferably be included in a modelling and simulation environment, since the model specification and the numerical integration of the model equations are integral parts of the method. The report Sørlie (1997) presents an interface between the modelling tool OmSim and the optimization tool IdKit. The case-study presented in Paper IV shows the benefits of such an integration. IdKit is also the basis in a model calibration and validation tool called MoCaVa, see Bohlin (1998). There is also ongoing work to implement MoCaVa in Matlab.

6

Conclusions

Modelling and simulation of dynamical systems is a difficult area for several reasons. It is not taught in a consistent way to engineering students, since it is an area of expertise which typically is scattered in the engineering faculty. Modelling and simulation also requires skills in many different subjects:

1. good knowledge of the application domain

2. understanding of the mathematical model

3. programming skills, on different levels depending on the tool used

4. numerical analysis, to correctly interpret results

Model libraries can provide support that helps inexperienced users attack complex problems. The support lies mainly within items 1. and 3. above, but also to some extent in the last since a correct model library can help avoiding numerical difficulties. Deeper understanding of mathematical models is mainly obtained by experience.

Modelling languages and simulation software tools on the other hand provide support for the two last items. The structure and syntax of the modelling language are important for ease-of-use and the quality of the models in a model library.

The main conclusion of this thesis is that modelling of complex systems should be supported by well-structured model libraries. Model libraries should be built using equation-based modelling. This is mainly because it relieves the user from the burden of manipulating equations/blocks into a computationally convenient form. But also because it reduces the number of model variants in the library. It is also shown how object-oriented modelling makes library maintenance easier. The task of symbolic manipulation is left to the simulation software, which of course needs to be more sophisticated.

The conclusions in the thesis are drawn from the development of two different model libraries. First, the simpler, more application-oriented library **K2** written in OMOLA. **K2** is intended for modelling of thermal power plants. Second, the more advanced library ThermoFlow, which covers general thermo-hydraulics and thus more application domains. ThermoFlow is written in Modelica and is a part of the Modelica base library package.

The thesis contains modelling examples of evaporation processes, both in drum-boilers and evaporation of continuous flow in a pipe. There is also a discussion on model validation methods for physical models.

6.1 Future Work

Continued library development

The definition of the thermo-hydraulic Modelica base library, ThermoFlow, is not the end of the story. The library covers all basic parts in thermo-hydraulic modelling, e. g., thermodynamic control volumes, momentum dynamics, bidirectional and two-phase flows. However, to build system models from these basic models, the user needs to know some thermodynamics. In many applications, there is a need for tailor-made component models, suited for the non-expert user. The base library gives rich opportunities to build such application libraries. However, design of application libraries is usually not research. This work should be left for model consulting companies, such as Sycon AB or Solvina, that can license application libraries to their customers. Models suited for specific purposes are usually not free of charge.

Uncertainty descriptions

Descriptions of parametric uncertainty is an area in linear systems theory that has grown rapidly during the last fifteen years, although the concept of uncertainty has existed much longer than that. For example Gille *et al.* (1959) proposed to capture the region of validity of a linear model with the notion of the *linearity lemon*, see Figure 6.1. This lemon shape could just as well be used to illustrate uncertainty.

One key idea of robust control theory is to capture uncertainty as additional blocks, see Figure 6.2. Uncertainty descriptions are often derived from comparing a more detailed, possibly nonlinear, system model to a simpler one that you want to work with in control design. To have tools that could extract a simple model for design from a complex physical

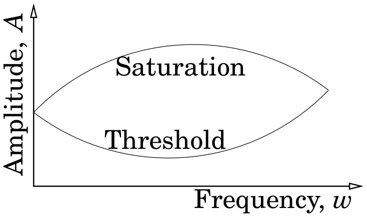


Figure 6.1 The linearity lemon, adopted from p.199 in Gille *et al.* (1959), describes the region of validity for linear transfer functions. The region is restricted from above by saturation phenomena, from below by threshold nonlinearities and at high frequencies by neglected dynamics.

model and also supply bounds for parametric uncertainty would be very powerful. Some tools like OMSIM and Dymola can automatically derive a linearized model from a nonlinear model. In Lantz and Rantzer (1998) OMSIM is used to export a linear model of a power network with parametric uncertainty. The model is exported on a form that can be used for analysis in the Matlab® μ-toolbox, see also Andersson (1999).

Another interesting possibility to get a bound on the amount of uncertainty in a model could be to estimate it from the model validation procedure. There are a large number of papers investigating this possibility, e. g., Rangan and Poolla (1998).

Model Analysis and Simplification

The focus in modelling has for a long time been on simulation. Today this is changing towards analysis and design. It would be useful to have tools that could extract simplified models from complex models, e. g., static relations, linearized models or low-order models with descriptions of parametric uncertainty. While doing manual model simplification you use know-

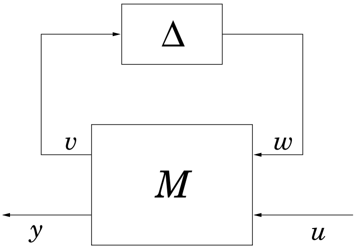


Figure 6.2 Uncertain system on uncertainty feedback form.

ledge that certain dynamics are on a different timescale and thus can be neglected, or that variables and terms in equations are of a smaller order of magnitude and can be removed from the model. Current work at the department explores how this kind of information can be extracted automatically from a model, e. g., Öhman (1998) gives results on model simplification using linearized models along a given trajectory. For this kind of model analysis it is useful to have both symbolic and numeric tools. Symbolic manipulation can give information about the structure of a problem. Analysis of the incidence matrix of an equation system shows where there are strong or weak couplings between different parts of the system, sometimes this information is not enough and then numeric values can be used.

7

References

Abelson, H. and G. J. Sussman (1985): *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA.

Andersson, L. (1999): *On Simplification of Models with Uncertainty*. PhD thesis ISRN LUTFD2/TFRT--1054--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Andersson, M. (1994): *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis ISRN LUTFD2/TFRT--1043--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Bauer, O. (1999): "Modelling of two-phase flows with Modelica." Master thesis ISRN LUTFD2/TFRT--5629--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Bell, R. D. and K. J. Åström (2000): "Drum-boiler dynamics." *Automatica*, **36:3**, pp. 363–378.

Bendotti, P. and C.-M. Falinower (1999): "EDF benchmark for robust control techniques – evaluation of proposed solutions." In *Preprints 14th World Congress of IFAC*, vol. G, pp. 455–460. Beijing, P.R. China.

Bohlin, T. (1998): "Process model calibrator and validator." In *Preprints of Reglermöte '98*, pp. 58–62. Lund Institute of Technology, Sweden.

Booch, G. (1991): *Object Oriented Design with Applications*. Benjamin/Cummings, Redwood City, California.

Butterfield, M. H. (1990): "A method of quantitative validation based on model distortion." *Trans. Inst. Measurement and Control*, **12:4**, pp. 167–173.

Butterfield, M. H. and P. J. Thomas (1986a): “Methods of quantitative validation for dynamic simulation models. I. Theory.” *Trans. Inst. Measurement and Control*, **8:4**, pp. 182–200.

Butterfield, M. H. and P. J. Thomas (1986b): “Methods of quantitative validation for dynamic simulation models. II. Examples.” *Trans. Inst. Measurement and Control*, **8:4**, pp. 201–219.

Cameron, R., R. L. Marcos and C. de Prada (1998): “Model validation of discrete transfer functions using the distortion method.” *Mathematical and Computer Modelling of Dynamical Systems*, **4:1**, pp. 58–72.

Cameron, R. G. (1992): “Model validation by the distortion method: linear state space systems.” *IEE Proceedings-D*, **139:3**, pp. 296–300.

Cellier, F. (1991): *Continuous System Modeling.* Springer-Verlag, New York, NY.

Chien, K. L., E. I. Ergin, C. Ling and A. Lee (1958): “Dynamic analysis of a boiler.” *Transactions of ASME*, **80**, pp. 1809–1819.

CVShome.org (2000): “Concurrent versions system.” www.cvshome.org.

Dullerud, G. E. and R. S. Smith (1997): “Invalidation techniques for assessing linear perturbation models of nonlinear systems.” In *Proceedings of the American Control Conference*, vol. 3, pp. 2078–2082.

Eborn, J. (1994): “Modelling and simulation of an industrial control loop with friction.” Master thesis ISRN LUTFD2/TFRT--5501--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Eborn, J. (1998a): “Experiences from using a model database for process modelling.” In *Preprints of Reglermöte ’98*, pp. 68–72. Lund Institute of Technology, Sweden.

Eborn, J. (1998b): *Modelling and Simulation of Thermal Power Plants.* Lic Tech thesis ISRN LUTFD2/TFRT--3219--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Eborn, J. and K. J. Åström (2000): “Modeling of a boiler pipe with two-phase flow instabilities.” In *Modelica 2000 Workshop Proceedings*, pp. 79–88. Modelica Association, Lund, Sweden.

Eborn, J. and B. Nilsson (1996): “Simulation of a thermal power plant using an object-oriented model database.” In *IFAC’96, Preprints 13th World Congress of IFAC*, vol. O, pp. 121–126. San Francisco, California.

Eborn, J. and H. Olsson (1995): "Modelling and simulation of an industrial control loop with friction." In *Proceedings of the 4th IEEE Conference on Control Applications*, pp. 316–322. Albany, New York.

Eborn, J., H. Panagopoulos and K. J. Åström (1999): "Robust PID control of steam generator water level." In *Preprints 14th World Congress of IFAC*, vol. G, pp. 461–464. Beijing, P.R. China.

Eborn, J., H. Tummescheit and F. Wagner (2000): "Development of a Modelica base library for modeling of thermo-hydraulic systems." In *Proceedings of the 41st SIMS Simulation Conference, SIMS'2000*, pp. 253–266. SIMS, Copenhagen, Denmark.

Eklund, K. (1971): *Linear Drum Boiler-Turbine Models*. PhD thesis TFRT-1001, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Elmqvist, H. (1972): "SIMNON – Ett interaktivt simuleringsprogram för olinjära system," (An interactive simulation program for nonlinear systems). Technical Report TFRT-5113. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Elmqvist, H., S. E. Mattsson and M. Otter (1999): "Modelica - a Language for Physical System Modeling, Visualization and Interaction." In *Proceedings of Symposium on Computer-Aided Control System Design, CACSD'99*. IEEE, Hawaii. Plenary paper.

Elmqvist, H. and M. Otter (1994): "Methods for tearing systems of equations in object-oriented modeling." In *ESM'94 European Simulation Multiconference*. Barcelona, Spain.

Gille, J.-C., P. Decaulne and M. Pelegrin (1959): *Feedback control systems*. McGraw-Hill, New York, NY.

Kant, I. (1781): *Kritik der reinen Vernunft*.

Klevhag, J. (1996): "Accuracy verification of continuous juice blending process using simulation." Master thesis ISRN LUTFD2/TFRT--5554--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Kwatny, H. G. and J. Berg (1993): "Drum level regulation at all loads." In *Preprints IFAC 12th World Congress*, vol. 3, pp. 405–408. Sydney, Australia.

Lantz, M. and A. Rantzer (1998): "Robustness analysis of large differential-algebraic systems with parametric uncertainty." In *Proceedings of MTNS98*. Padova, Italy.

Leva, A., C. Maffezzoni and G. Benelli (1999): "Validation of drum-boiler models through complete dynamic tests." *Control Engineering Practice*, **7:1**, pp. 11–26.

Lindahl, S. (1976): *Design and Simulation of a Coordinated Drum Boiler-Turbine Controller*. Lic Tech thesis TFRT-3143, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Ljung, L. (1987): *System Identification—Theory for the User*. Prentice Hall, Englewood Cliffs, New Jersey.

Ljung, L. and L. Guo (1997): "Classical model validation for control design purposes." *Mathematical Modelling of Systems*, **3:1**, pp. 27–42.

Ljung, L. and H. Hjalmarsson (1995): "System identification through the eyes of model validation." In *Proceedings of the Third European Control Conference, ECC'95*, vol. 2, pp. 949–954. Rome, Italy.

Löfgren, O. and P. Svensson (1997): "Modelling and control of a plate heat exchanger in steam applications." Master thesis ISRN LUTFD2/TFRT--5584--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Maciejowski, J. M. (2000): *Predictive Control with Constraints*. Addison-Wesley, Wokingham, England.

Mattsson, S. E. (1995): "Simulation of object-oriented continuous time models." *Mathematics and Computers in Simulation*, **39**, pp. 513–518.

Mattsson, S. E., H. Olsson and H. Elmqvist (2000): "Dynamic selection of states in dymola." In *Modelica 2000 Workshop Proceedings*, pp. 61–67. Modelica Association, Lund, Sweden.

Mattsson, S. E., M. Otter and H. Elmqvist (1999): "Modelica hybrid modeling and efficient simulation." In *38th IEEE Conference on Decision and Control, CDC'99*. Phoenix, Arizona.

Modelica Design Group (2000): "The Modelica Language Specification." Version 1.4, `http://www.modelica.org/`.

Murray-Smith, D. J. (1998): "Methods for the external validation of continuous system simulation models: a review." *Mathematical and Computer Modelling of Dynamical Systems*, **4:1**, pp. 5–31.

Nilsson, B. (1993): *Object-Oriented Modeling of Chemical Processes*. PhD thesis ISRN LUTFD2/TFRT--1041--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Nilsson, B. and J. Eborn (1995): “An object-oriented model database for thermal power plants.” In Breitenecker and Husinsky, Eds., *Eurosim ’95 Simulation Congress*, pp. 747–752. Elsevier.

Öhman, M. (1998): *Trajectory-Based Model Reduction of Nonlinear Systems*. Lic Tech thesis ISRN LUTFD2/TFRT--3223--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Ordys, A., R. Katebi, M. Johnson and M. Grimble (1994): *Modelling and Simulation of Power Generation Plants*. Springer-Verlag, London, UK.

Paynter, H. (1961): *Analysis and Design of Engineering Systems*. MIT Press, Cambridge, MA.

Pfafferott, T. and G. Schmitz (2000): “Numeric simulation of an integrated CO_2 cooling system.” In *Modelica 2000 Workshop Proceedings*, pp. 89–92. Modelica Association, Lund, Sweden.

Pontremoli, A. (2000): “Modeling and control of a paper dryer section using modelica.” Master thesis ISRN LUTFD2/TFRT--5653--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Popper, K. R. (1935): *Logik der Forschung*. Julius Springer, Vienna, Austria.

Ramos González, J. J. (1994): “Object-oriented modelling of flows in process systems.” Report ISRN LUTFD2/TFRT--7521--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Rangan, S. and K. Poolla (1998): “Model validation for structured uncertainty models.” In *Proceedings of the 1998 American Control Conference*, vol. 1, pp. 629–633. Philadelphia, Pennsylvania.

Rumbaugh, J. (1991): *Object-oriented modeling and design*. Prentice-Hall International, Englewood Cliffs, New Jersey.

Smith, R. S., G. E. Dullerud, S. Rangan and K. Poolla (1997): “Model validation for dynamically uncertain systems.” *Mathematical Modelling of Systems*, **3:1**, pp. 43–58.

Sørlie, J. (1997): “On an interface between OmSim and IdKit.” Report ISRN LUTFD2/TFRT--7562--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Sørlie, J. and J. Eborn (1997): “A grey-box identification case study: The Åström–Bell drum-boiler model.” Technical Report ISRN LUTFD2/TFRT--7563--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Sørlie, J. and J. Eborn (1998): “Parameter optimization results for a family of thermo-physical drum boiler models.” In *Preprints of Reglermöte ’98*, pp. 131–136. Lund Institute of Technology, Sweden.

Stojnic, P. (1997): “Modeling of steam generation in a sulphuric acid plant.” Technical Report ISRN LUTFD2/TFRT--5577--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Strömberg, J.-E. (1994): *A mode switching modelling philosophy*. PhD thesis 353, Dept. of Electrical Engineering, Linköping University, Linköping, Sweden.

Tummescheit, H., J. Eborn and F. Wagner (2000): “Development of a Modelica base library for modeling of thermo-hydraulic systems.” In *Modelica 2000 Workshop Proceedings*, pp. 41–51. Modelica Association, Lund, Sweden.

Ward, J. and R. Middleton (1999): “Sequential approach to control systems synthesis with constraints.” In *Preprints 14th World Congress of IFAC*, vol. G, pp. 483–488. Beijing, P.R. China.

Paper I

Object-Oriented Modelling of Thermal Power Plants

Bernt Nilsson and Jonas Eborn

Abstract

This paper presents a set of model libraries, called K2, for modelling of thermal power plants. The models are based on first principles and describe mainly the dynamic mass and energy properties of the modelled system. The K2 models are described in the object-oriented modelling language OMOLA and the libraries are organized in an OMOLA model database. The libraries are grouped into three different sets, namely unit libraries, subunit libraries and model component libraries. The unit models are used to build up plant system models, which are application dependent. The units are composed of subunits. The subunits describe different physical phenomena and a set of subunits build up the behaviour of the unit model. Model components are used to facilitate the development of new units and subunits. OMOLA models can be simulated in the OMSIM simulation environment and the K2 model database has been used in a case study of the dynamics in an HRSG plant.

Keywords: Computer aided engineering, modelling, object-oriented modelling, power plant, process models.

The full paper can not be reproduced here for copyright reasons. It is available (by subscription) from

Mathematical and Computer Modelling of Dynamical Systems, **4:3**, pp. 207–218.

Paper II

Development of a Modelica Base Library for Modeling of Thermo-hydraulic Systems

Jonas Eborn, Hubertus Tummescheit and Falko Jens Wagner

Abstract

This paper presents current results of an ongoing project to develop a Modelica base library for thermo-hydraulic systems. There are many different aspects to the development of such a library, from the basic physics of fluids and heat to the structuring of model classes in the library and the actual implementation in the Modelica language. The structuring should define interfaces and partial classes that facilitate reuse to make the library general and easy to use. Different choices of media, use of different state variables as well as different levels of complexity in modeling is anticipated in the library structure.

The basic entity in the library is the model of a control volume. It is formed by multiple inheritance from three parts; the partial thermal model, the partial hydraulic model and the medium property model. *Flexibility* is obtained by parameterizing this control volume. It can be either lumped or discretized in *n* sections along the flow direction. The three parts are also parameterized with class parameters. This means for example that you can easily exchange the medium properties in a control volume.

The aim of the project is to develop a model library that contains all basic components needed for thermo-hydraulic systems. Besides control volumes and medium models this also means models for simple machinery, e. g., pumps, valves and heat exchangers. Code examples are given in the paper.

Extended version of

Eborn, J., H. Tummescheit, and F.J. Wagner (2000): "Development of a Modelica base library for modeling of thermo-hydraulic systems." In *Proceedings of the 41st SIMS Simulation Conference, SIMS'2000*, pp. 253–266. SIMS, Copenhagen, Denmark.

1. Introduction

With the modeling language Modelica™, Elmqvist *et al.* (1999); Modelica Design Group (2000), it is possible to create model libraries for different application areas. In the Modelica base library distribution there are libraries for multi-body systems, electrical systems and block diagrams. A base library for modeling and simulation of thermo-hydraulic systems is also necessary to further expand the range of applications for Modelica. A thermo-hydraulic base library should cover the basic physics of flows of fluids and heat. It also needs to cover models for properties of fluids like water and refrigerants. The library would then be useful in several application areas, e. g., power generation plants, district heating and refrigeration systems.

The general goal of the library is to provide a framework and basic building blocks for modeling thermo-hydraulic systems in Modelica. For obvious reasons it is impossible to provide components for every application, so one of the main goals is extensibility. For the same reason, much more emphasis will be put on the basic parts of the library, such as medium models and essential control volumes, than on an exhaustive application library. The focus of the library is on models of homogeneous one- and two-phase flows, non-homogeneous and multi-phase flows are not taken into account yet. It is necessary to support bidirectional flow, because flow directions can change during simulation or are not known initially in networks.

To make the library general and extensible, the design must accommodate for example different choices of media, single/multi-component flow and one- or two-phase flows. For numerical efficiency, it may also be interesting to use different pairs of state variables, e. g., $\{p, h\}$, $\{p, T\}$ or $\{\rho, T\}$. This is anticipated in the library structure. The basic entity in the library, the control volume, is built up by multiple inheritance from three parts. The partial thermal model contains dynamic state equations derived from conservation laws of mass and energy. The partial hydraulic model contains the mass flow equation that is formed from either a static or a dynamic momentum balance. The third part in the control volume is the medium model that calls the appropriate medium property functions.

The models in the library are designed for system level simulation, not for detailed simulation of flows, which are usually done in CFD packages. The models are thus discretized in one dimension or even lumped parameter approximations.

It has to be emphasized that, especially in the area of fluid flow, different assumptions about the importance of terms in the general equations can lead to models which are very different mathematically. The library offers only a limited selection of assumptions, which nonetheless should cover a broad range of applications.

Some basic ideas for the thermo-hydraulic base library have previously been presented in Tummescheit and Eborn (1998); Eborn *et al.* (1999b); Tummescheit (2000); Tummescheit (2000b). Object-oriented component based modeling has been presented in Wagner and Poulsen (1999); Wagner *et al.* (1998).

2. Basic ideas

The basic design principles of the library are:

- one unified library both for lumped and distributed parameter models,
- both bi- and unidirectional flows are supported,
- separation of the medium submodels, which can be selected through class parameters,
- assumptions (e.g., gravity influence yes or no) can be selected by the user from the user interface.

The first guideline puts a constraint on the discretization method used in the distributed parameter models: only the so-called "staggered grid" method, see Figure 1, reduces to a useful model in the lumped parameter case. In this method, Harlow and Welch (1965), all fluxes are calculated on the border of a control volume and the intensive quantities are calculated in the center of a control volume. The method is a special case of the finite volume method, Patankar (1980), which is common for systems modeling with one-dimensional discretizations.

The ability to handle reversing flows requires extra information in the connectors between models. Transported properties, e.g., enthalpy and composition, would need to be included twice, upstream and downstream.

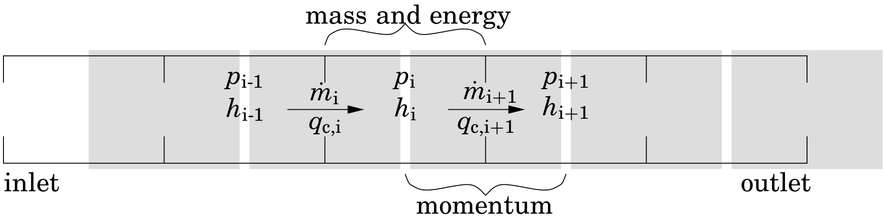


Figure 1 Staggered grid discretization. The control volumes for mass/energy and momentum are translated relative to each other.

This has instead been solved by including convective heat flow and component mass flows in the connectors. Thus the information needed for the balance equations (see Section 3) is contained in variables depending on the flow direction, i. e., mass flow and convective heat flow. In contrast the transported properties in the connector are always taken from the closest control volume.

The connector for single medium flow without dynamic momentum balance then contains the variables:

$$\{p, h, \dot{m}, q_c, \rho, T, s, \kappa\} \tag{1}$$

where the quantities are pressure, specific enthalpy, mass flow, convective heat flow, density, temperature, specific entropy and ratio of specific heats, respectively.

The different kinds of lumped and distributed models are shown in Figure 2. With this choice the user has the possibility to change the complexity of a system model. The staggered grid is reflected in the splitting into different lumped model types, volume models and flow models. Different appearance of the connectors ensure the alternating structure, inherent in the staggered grid method. Always connecting different connectors guarantees that the simulation problem is well specified and that there are no unnecessary algebraic loops.

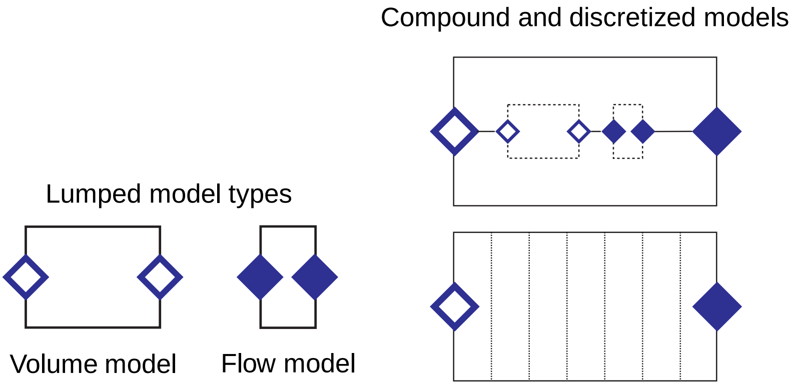


Figure 2 Lumped, compound and discretized model objects. Filled and outlined connectors are different to ensure an alternating connection structure.

3. Control volume equations

The basic thermodynamic equations governing a fluid system are partial differential equations. In our discretized setting these are integrated over a fixed control volume to obtain ordinary differential equations.

For a complete model description of a control volume, four parts are combined:

- balance equations (mass, energy and momentum),
- constitutive equations (pressure drop, heat flow),
- medium properties,
- transformations of state variables.

Each of these parts are explained in the following sections. The three balance equations give three of the variables in (1), the remaining are obtained from medium property routines. Below we give the basic form of the balance equations. Usually the balance equations in mass and energy are not used directly, but transformed into some pair from $\{\{p, \rho\}, \{h, T, s, u\}\}$ for numerical efficiency reasons. This transformation is explained in the last section below.

Balance equations

With the staggered grid approximation described in Section 2 the balance equations are split up. The volume model holds the equations for total mass and internal energy. If there are n flow connections to other volumes and l heat transfer areas these are written as (positive flow into the volume):

$$\frac{d}{dt}\begin{pmatrix} M \\ U \end{pmatrix} = \begin{pmatrix} \sum_i^n \dot{m}_i \\ \sum_i^n q_{c,i} + \sum_j^l q_{\text{transfer},j} \end{pmatrix} \tag{2}$$

Between the volumes there must be a flow model, which holds the momentum balance. There are two types of flow models implemented in the library:

- Stationary pressure drop model
- Dynamic momentum balance for pipes with constant cross-sectional area.

Static flow models are much used in system simulation where the thermal behavior is the main concern. The dynamic momentum balance is useful for pressure wave propagation studies in a system which is mainly modeled with distributed models. Keep in mind that it is possible to add

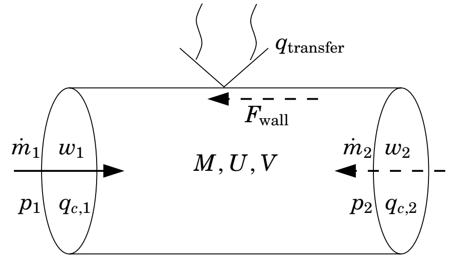


Figure 3 Schematic of a pipe section, used as a thermodynamic control volume. The arrows indicate reference direction for the flow quantities mass flow $\dot{m}$, velocity w and heat flow q.

other types of flow models to the existing structure, e. g., a momentum balance for variable cross-sectional area along the flow channel.

The dynamic momentum balance is derived from Newton's second law, applied to a pipe section with volume V and constant cross-section area A,

$$\frac{d}{dt}\int_V \rho\mathbf{w}\,dV = -\int_{A_{1,2}} \rho\mathbf{w}(\mathbf{wn})\,dA - \int_{A_{1,2}} p\mathbf{n}\,dA + \int_{A_w} \mathbf{Tn}\,dA + \int_V \rho\mathbf{g}\,dV$$

where vector quantities are marked with boldface symbols and $\mathbf{T}$ is the stress tensor. The left hand side is the time derivative of the momentum in the volume V. The first term on the right hand side is the momentum flux over the end surfaces, the second term are pressure forces acting on the end surfaces, the third term are shear stress acting on the pipe wall and the last term is gravitational forces. In the axial direction, $\mathbf{e}_z$, we have $\mathbf{n}_1\mathbf{e}_z = -1$, $\mathbf{n}_2\mathbf{e}_z = 1$ and $\mathbf{w} = w\mathbf{e}_z$. By integrating over the control volume and using $\dot{m} = A\rho w$, we obtain a differential equation for mass flow rate,

$$\begin{aligned} I &= \int_V \rho w\,dV = \int_{\Delta z}\int_A \rho w\,dAdz = \dot{m}\Delta z \\ G_i &= \int_{A_i} \rho w^2\,dA = \dot{m}_i^2/A\rho_i \\ \Delta z\frac{d\dot{m}}{dt} = \frac{dI}{dt} &= G_1 - G_2 + (p_1 - p_2)A - F_{\text{wall}} - F_{\text{grav}} \end{aligned} \tag{3}$$

where G is the momentum flux. The wall friction force, F_{wall}, is given by some constitutive equation and the gravitational force, F_{grav}, depends on the angle against the plane.

Constitutive equations

The constitutive equations are empirical relations for heat flow, pressure drop and characteristics of machinery. They are typically formulated as characteristic equations for individual components, often algebraic equations but they could also be formulated as differential equations. For example in a pump, there exist many different relationships between mass flow rate, pressure increase, angular speed and consumed power. The fluid flow literature also holds many different expressions for the relationship between flow and pressure drop depending on the flow regime, pipe parameters, etc.

These constitutive equations should be replaceable, in order to have a general model for a component that can be used in different situations by exchanging the model for the characteristics, see Section 4 for an example.

Medium property routines

For simulation of thermo-hydraulic systems, it is necessary to have accurate models for the thermodynamic properties of the fluid that is flowing in the system. For the purpose of dynamic system simulation, the following criteria have to be met:

- Accuracy
- Speed
- Robustness

In some areas there exist recommended formulations (IAPWS/IF97 for water, Wagner and Kruse (1998)) or de facto standards (NIST-REFPROP routines for refrigerants, McLinden *et al.* (1998)) that have to be taken into account. External function call interfaces in Modelica make it possible to use these standards directly. Available routines and most medium property models in the literature (see, e.g., Reid *et al.* (1987)) are designed with stationary calculations in mind, therefore they have to be extended to include some needed extra derivatives for dynamic calculations.

In dynamic simulations the speed of the medium property functions is very important for the performance of the simulations. Whenever possible the medium properties should be non-iterative, which is the case when they are explicit in the dynamic states. This is easy to achieve for the steam tables, where the industrial standard formulation, IAPWS-IF97, has explicit routines for a variety of input variables (pressure and temperature, enthalpy or entropy). The complete industrial steam tables are implemented in the library.

For other properties, e.g. refrigerant R134a, such inverse formulations are not available. However, it is still possible to save a huge amount of computation time by precomputing the phase boundaries off-line and use an auxiliary equation for it. These vapor-liquid equilibrium calculations (VLE) for cubic and other medium models have to be performed iteratively and numerically, either by using Maxwell's criterium or calculating that Gibbs' free enthalpy is equal for both phases. The numerical calculations are too inefficient to be performed at each time step during dynamic simulation. In order to calculate medium properties inside the two-phase region, it is for non-transient states sufficient to know the properties on the phase boundaries and interpolate with the vapor mass fraction x. An efficient implementation of medium properties for pure components requires that VLE are calculated before the simulation and that VLE data is approximated either with a suitable function or with smooth spline interpolation. For the above listed media, high accuracy approximations are either available in the standard formulation (e. g., partially for water and CO_2) or provided in the base library.

The phase boundaries require special attention: the derivatives of most properties are discontinuous across the phase transition and therefore this has to be implemented as a discrete change which restarts the integration routine if a control volume changes its phase. This is a robustness requirement for most normal cases, but it can lead to unexpected "sliding mode" behavior, if e. g., heat transfer coefficients also change discontinuously at the phase boundary.

Currently we have implemented high-accuracy medium models for the whole fluid region for water, carbon dioxide and R134a. More refrigerant properties will be available soon. It is relatively easy to add new medium models to the existing ones and it is even simpler to exchange the medium model in existing models against another one.

To summarize, the medium properties that are provided with this library:

- are adapted for use with dynamic simulations.
- use non-iterative, auxiliary equations for the calculation of VLE.
- are highly accurate for water, CO_2 and $R134a$.
- include ideal gas properties for a wide variety of gases.

State variable transformations

There is an interdependence between the choice of the medium model and the selection of state variables. Many details of the medium model depend on the choice of the state equations. Most medium models are available for all of the choices of state variables in the library, but the numerical efficiency can be very different. The common choice $\{p, h\}$ is very efficient for water in the two-phase region where the IF97 medium model is explicit in these states, while it is slower at super-critical pressures, since the medium model is explicit in $\{\rho, T\}$ and thus iterations are needed.

The balance equations for mass and internal energy (2) can be rewritten into differential equations for ρ and u. A differentiation of $M = \rho V$ and $U = uM$ for a constant volume yields:

$$\left\{\begin{aligned} V\frac{d\rho}{dt} &= \frac{dM}{dt} \\ M\frac{du}{dt} &= \frac{dU}{dt} - u\frac{dM}{dt} \end{aligned}\right. \tag{4}$$

These primary equations are then transformed into secondary forms to give differential equations in the states suitable with the medium model. For example, if pressure and enthalpy are chosen as states,

$$\frac{d}{dt}\begin{pmatrix} \rho \\ u \end{pmatrix} = \underbrace{\begin{pmatrix} \left.\dfrac{\partial\rho}{\partial p}\right|_h & \left.\dfrac{\partial\rho}{\partial h}\right|_p \\ \left.\dfrac{\partial u}{\partial p}\right|_h & \left.\dfrac{\partial u}{\partial h}\right|_p \end{pmatrix}}_{\text{Jacobian, J}} \frac{d}{dt}\begin{pmatrix} p \\ h \end{pmatrix} \tag{5}$$

To obtain differential equations for pressure and enthalpy (5) must be solved for the derivative of (p, h)

$$\frac{d}{dt}\begin{pmatrix} p \\ h \end{pmatrix} = \mathsf{J}^{-1}\frac{d}{dt}\begin{pmatrix} \rho \\ u \end{pmatrix} \tag{6}$$

The partial derivatives of ρ are calculated in the medium model, while the partial derivatives of u can be reduced to those of ρ. From $u = h - p/\rho$ we obtain

$$\left.\frac{\partial u}{\partial h}\right|_p = 1 + \frac{p}{\rho^2}\left.\frac{\partial\rho}{\partial h}\right|_p \qquad \left.\frac{\partial u}{\partial p}\right|_h = -\frac{1}{\rho} + \frac{p}{\rho^2}\left.\frac{\partial\rho}{\partial p}\right|_h$$

This gives the inverse Jacobian as

$$\mathsf{J}^{-1} = \frac{a^2}{\rho}\begin{pmatrix} \rho + \dfrac{p}{\rho}\left.\dfrac{\partial\rho}{\partial h}\right|_p & -\rho\left.\dfrac{\partial\rho}{\partial h}\right|_p \\ 1 - \dfrac{p}{\rho}\left.\dfrac{\partial\rho}{\partial p}\right|_h & \rho\left.\dfrac{\partial\rho}{\partial p}\right|_h \end{pmatrix}$$

where a is the velocity of sound. By combining (6) and (4), multiplying with $M = \rho V$ and noting that $h = u + p/\rho$ we obtain

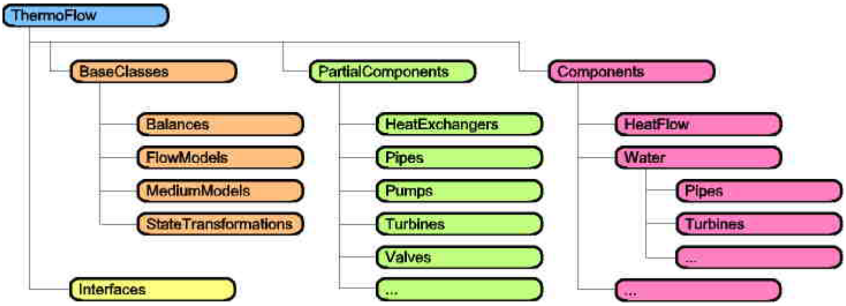


Figure 4 Package structure of the ThermoFlow library

$$\left\{\begin{aligned} V\frac{\rho}{a^2}\frac{dp}{dt} &= \left(\rho + h\left.\frac{\partial\rho}{\partial h}\right|_p\right)\frac{dM}{dt} - \left.\frac{\partial\rho}{\partial h}\right|_p\frac{dU}{dt} \\ V\frac{\rho}{a^2}\frac{dh}{dt} &= \left(1 - h\left.\frac{\partial\rho}{\partial p}\right|_h\right)\frac{dM}{dt} + \left.\frac{\partial\rho}{\partial p}\right|_h\frac{dU}{dt} \end{aligned}\right.$$

which are the differential equations for p, h used in ThermoFlow. Similar expressions have also been derived for other pairs of state variables, for example $\{p, T\}, \{p, s\}$ or $\{\rho, T\}$.

Library structure

The main idea of the ThermoFlow library is to provide an extensible basis for a robust thermo-hydraulic component library. The structure of the library is divided into four parts, see Figure 4.

Interfaces define the types of connectors used in the library. The flow connectors are of four different types; either single or multi-component flow medium, with a static or dynamic flow description. This package should be on the top level, since it is important for the interoperability between different Modelica base libraries.

Base classes are the central part of models, the basic physical equations for a control volume: balance equations, state transformations and medium models.

Partial components contain common expressions for component models, this allows code sharing and simplifies maintenance.

Components are the user part of the library, models that can be used to build a system for simulation.

4. Object-oriented modeling

Modelica is an object-oriented modeling language, designed for modeling physical systems. Many of the object-oriented features defined by Abadi and Cardelli (1996) are found in the Modelica language:

- (Multiple) Inheritance
- Class parameterization
- Generalization

The concept of inheritance lets one object inherit methods and properties (i. e., the behavior) from other objects. This allows code sharing and calls for applying generalization.

Class parameterization gives the possibility of building generic classes that can be used for specialization later. With this, a parameter can be passed to a class during instantiation, giving the class the desired behavior, see the examples below.

In a way, the concept of object-orientation, in relation to component based modeling, inspires the user to generalize the system. This can lead to a better understanding of the system to be modeled. Through generalization, the user is forced to decompose the system into parts. Each part can then be modeled and implemented in meaningful classes. These classes tend to represent the essential parts of the system, and aggregation (through multiple inheritance) collects these parts again to form a complete model of the system.

Object-oriented constructs in Modelica

In the following subsections we will give examples of how the object-oriented features described above are handled in Modelica.

Aggregation through multiple inheritance is used to build up basic models. A control volume formulation of a pipe can be decomposed in the following individual parts:

- Balance equations
- Flow model
- A shell model, which defines the connectors

These parts are modeled individually in the following classes:

```
partial model Balances
  ... some equations;
end Balances;
```

```
  partial model FlowModel
    ... some equations;
  end FlowModel;
  partial model TwoPort
    FlowConnector a,b;
  end TwoPort;
```

and aggregation of these base classes leads to a general description of a control volume, e. g., a pipe

```
model Pipe
  extends TwoPort;
  extends Balances;
  extends FlowModel;
end Pipe;
```

By the Modelica keyword **extends** the new model Pipe inherits all attributes of the base classes. Common parts of the base classes are only inherited once.

Class parameterization is used to add replaceable objects to a class. This object can then be replaced by passing a specific class as a parameter during instantiation.

As an example we take the FlowModel from above. Any flow model needs some type of pressure loss model. In order to make the class FlowModel as general as possible, we only specify a generic flow model during base class implementation.

```
partial model FlowModel
  replaceable class
    Ploss = GenericPressureLossModel;
  extends Ploss;
end FlowModel;
```

The GenericPressureLossModel does not have to contain any equations, but for practical reasons (and as a base class for inheritance in specialized pressure loss models) it contains the variables that are common to all pressure loss models.

In specialized classes, the generic pressure loss model is then replaced by a more meaningful model, which also contains equations for calculating the actual pressure loss.

```
model SpecialPipe
  extends TwoPort;
  extends Balances;
  extends FlowModel(redeclare
    Ploss = SpecialPressureLossModel);
end SpecialPipe;
```

Generalization is the key element in object-orientation. It is closely related to the notion of *classes*. A class describes some general behavior of objects that have some properties in common. Exactly these common properties call for a general description - a class. The purpose is obviously code sharing, but an often quite appreciated side effect of this is a better understanding of the problem being modeled.

Generalization is in this library used to specify behavior of components, which for some reason is common to all components of that particular type. For flow equipment a general feature is the convective heat transport, which can be expressed as

```
partial model FlowModelBase
  extends FlowVariables;
  extends TwoPort;
equation
  a.q_conv = if a_upstr
    then mdot*a.h
    else mdot*b.h;
end FlowModelBase;
```

where the specification of the flow direction, a_upstr, and the mass flow, mdot, is postponed until later.

Since the calculation of the mass flow depends on the type of flow equipment used, this additional information has to be provided in a specialized class. For example a flow resistance with a linear expression for pressure losses

```
model LinearValve
  extends FlowModelBase;
equation
  a_upstr = a.p > b.p;
  mdot = mdot0/dp0*(a.p-b.p);
end LinearValve;
```

where the mass flow depends on the parameters mdot0, dp0 and the pressure difference over the valve.

Summary

Some examples have shown how important object-oriented constructs of the Modelica language are implemented in ThermoFlow. The constructs are used throughout the library structure (see Section 3) to facilitate wide spread use of generalization and code sharing and make the library more flexible.

5. Component models

As mentioned earlier, the aim of this project was not extensive component modeling, but to create a base structure for future development of component models. For demonstration purposes a few component models have been implemented. This section presents some of them.

Pumps

For modeling a pump, e.g., feed water pump, it is necessary to have a relationship between the volume flow rate, the pressure increase and the speed of the pump. This is called the pump characteristic or pump profile. One example of an expression for this relationship is

$$\Delta p_n = R_1 n_n + 2R_2 n_n V_n - R_3 \left| V_n \right| V_n \tag{7}$$

where, p_n, n_n and V_n are normalized variables for pressure p, speed of the pump n and volume flow rate V. The design point is $(p_n, n_n, V_n) = (1, 1, 1)$ and represents the pump in normal operation.

In terms of the ThermoFlow library structure, the pump can be modeled by using a lumped control volume and a lumped flow model according to Figure 2. The lumped flow model then represents the pump characteristic (7), whereas the lumped control volume before the pump is used according to the library structure. This control volume represents the volume of the pump, which should not be neglected.

Heat exchangers

A heat exchanger is modeled using base components from the library. Basically, it consists of two pipes connected by a heat conducting wall. Figure 5 shows the principle of modeling a heat exchanger and a sample system.

Heat exchangers can either be lumped or distributed. In the distributed case the heat transfer model uses a simple temperature difference between the individual elements of the distributed pipes. The lumped case is either based on this simple model or uses the logarithmic mean temperature. Furthermore, tube and shell heat exchangers are implemented using a circular wall geometry with an inner and an outer pipe.

Turbines

Analogous to pumps, a turbine is modeled as a lumped control volume with a following lumped flow model. The flow model introduces the turbine characteristic or turbine profile. Currently two models are implemented in the base library. One model is according to Stodola, see Stodola (1927).

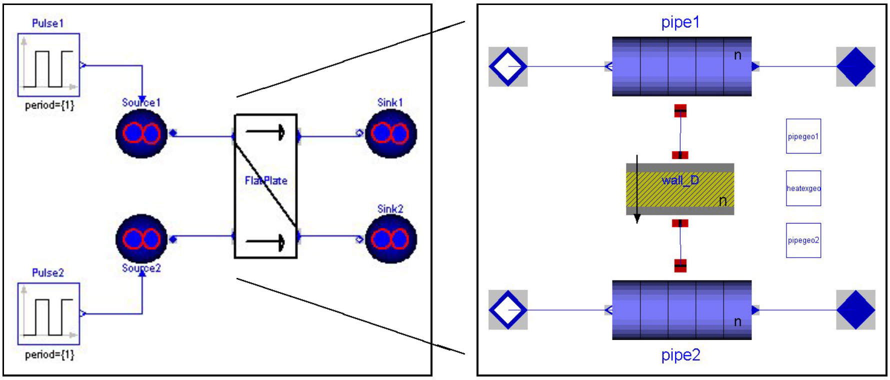


Figure 5 Example system using a heat exchanger consisting of two pipes, a wall and 4 connectors

The Stodola model is an idealized turbine with an infinite number of stages. The other model is due to Linnecken, see Traupel (1977). The Linnecken model considers the maximum mass flow rate through the turbine stage and can be parameterized according to the type of the turbine and the number of stages.

Reservoirs

Some thermo-hydraulic systems are closed systems, e.g., power plants or refrigeration systems. But for general modeling purposes, sources and sinks are required. These are used to add "boundary" conditions to other components or systems of components. Typical sources are temperature, pressure or heat sources. They can either be fixed or depend on some input signal. The sample system in Figure 5 contains 2 controlled sources and 2 sinks.

For flow sources, the model consists of a control volume, giving thermodynamic properties to the supplied flow, and a flow model at the outlet of the source. The control volume is a so called "infinite reservoir", i.e., the thermodynamic conditions do not vary over time as mass and energy leaves or enters the control volume. The flow model is used to make the sources more realistic (and numerically less stiff), e.g., by modeling a pressure drop over the outlet. A flow sink is just a fixed pressure control volume, and the assumption about the infinite reservoir holds as well.

Heat sources can give either a fixed temperature or a fixed heat flow. They can also be controlled by an external signal.

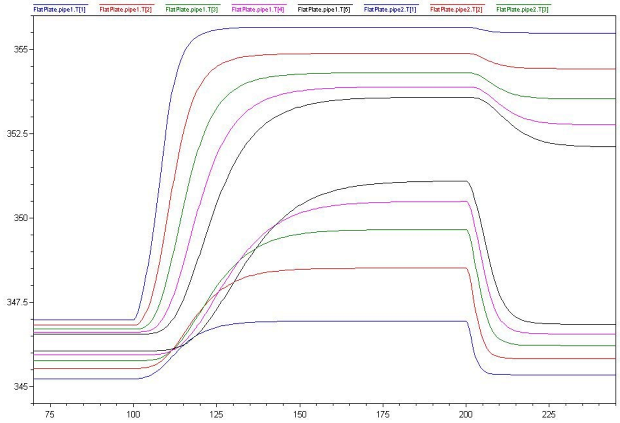


Figure 6 Resulting temperatures from simulating the system in Figure 5

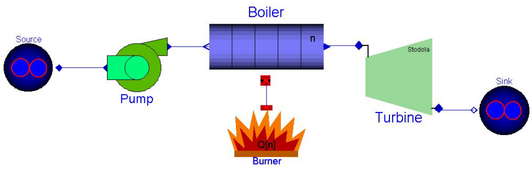


Figure 7 Example system

6. Examples

A system with a parallel flow plate heat exchanger (see Figure 5) is simulated with a temperature increase on the hot side, followed by a pressure increase on the cold side with according mass flow rate increase. The result is shown in Figure 6.

What can be seen from the results is that an increase in the temperature on the hot side also increases the temperature on the cold side. Since this is a parallel flow heat exchanger, the temperature difference between the hot and the cold side also increases. The following flow increase on the cold side causes the temperature difference between the hot and the cold side to increase, and the temperature on the hot side drops accordingly due to the increase in the heat flow to the cold side.

Using the components implemented in the library, it is possible to build also more complex systems, e. g., power plants, see Figure 7.

7. Conclusion

In the design of the base library, the concepts of object-oriented modeling have been used to make the library flexible and easy to use. The generalization splits a complex problem into subproblems, which are modeled individually (e. g., balance equations, momentum equations, heat transfer) and aggregated to build component models. This separation simplifies library maintenance and makes building many model variants easier.

The Modelica language offers standard object-oriented features, such as composition and inheritance as well as more advanced features like class parameterization. Using these, the basic constraints from thermo-hydraulic modeling are inherent in the library models, but they can still be made flexible and extensible through specialization and class parameterization. The decomposition of models sometimes makes it difficult to get an overview over inherited parts of a model. However, the advantages with a more maintainable structure and reusable classes outweigh this disadvantage.

Some further conclusions:

- **Ease of use:** Taking the user perspective early in the library design process is important for the final result.
- **Nomenclature of research field:** Use of known symbols is very important for the usefulness of the library.
- **No overkill:** There is a risk of over-structuring using object-oriented methods.

We have also seen, that it is possible to model complex systems with the components implemented in the library. The modeling and simulation tool Dymola™ has been used in the design of the library. Dymola has a graphical user interface that allows drag and drop model editing, making the modeling process easier.

Please note, because of the structure of the library, only verification of the base models is possible. Real model validation is only possible in a system context, which has been done for a few examples. Also, the library is meant as a basis for further development. Models of basic control volumes and flow models are complete, but there is a need for many more components for different application areas.

Further information

For the interested reader, further information about the ThermoFlow project can be obtained at www.control.lth.se/˜hubertus/ThermoFlow or by contacting the authors.

References

Abadi, M. and L. Cardelli (1996): *A Theory of Objects*. Springer, New York, Berlin.

Eborn, J. (1998): "Experiences from using a model database for process modelling." In *Preprints of Reglermöte '98*, pp. 68–72. Lund Institute of Technology, Sweden.

Eborn, J. and B. Nilsson (1996): "Simulation of a thermal power plant using an object-oriented model database." In *IFAC'96, Preprints 13th World Congress of IFAC*, vol. O, pp. 121–126. San Francisco, California.

Eborn, J. and H. Olsson (1995): "Modelling and simulation of an industrial control loop with friction." In *Proceedings of the 4th IEEE Conference on Control Applications*, pp. 316–322. Albany, New York.

Eborn, J., H. Panagopoulos, and K. J. Åström (1999a): "Robust PID control of steam generator water level." In *Preprints 14th World Congress of IFAC*, vol. G, pp. 461–464. Beijing, P.R. China.

Eborn, J., H. Tummescheit, and K. J. Åström (1999b): "Physical system modeling with Modelica." In *14th World Congress of IFAC*, vol. N. IFAC.

Elmqvist, H., S. E. Mattsson, and M. Otter (1999): "Modelica - a Language for Physical System Modeling, Visualization and Interaction." In *Proceedings of Symposium on Computer-Aided Control System Design, CACSD'99*. IEEE, Hawaii. Plenary paper.

Elmqvist, H. and M. Otter (1994): "Methods for tearing systems of equations in object-oriented modeling." In *ESM'94 European Simulation Multiconference*. Barcelona, Spain.

Harlow, F. H. and J. E. Welch (1965): "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface." *Phys. Fluids*, **8**, pp. 2182 – 2189.

Mattsson, S. E. (1995): "Simulation of object-oriented continuous time models." *Mathematics and Computers in Simulation*, **39**, pp. 513–518.

Mattsson, S. E. and M. Andersson (1992): "The ideas behind Omola." In *Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design, CACSD '92*. Napa, California.

Mattsson, S. E., M. Otter, and H. Elmqvist (1999): "Modelica hybrid modeling and efficient simulation." In *38th IEEE Conference on Decision and Control, CDC'99*. Phoenix, Arizona.

McLinden, M. O., S. A. Klein, E. W. Lemmon, and A. P. Peskin (1998): *NIST Thermodynamic and Transport Properties of Refrigerants and Refrigerant Mixtures—REFPROP*. U.S. Department of Commerce, version 6.0 edition.

Modelica Design Group (2000): "The Modelica Language Specification." Version 1.4, `http://www.modelica.org/`.

Nilsson, B. (1994): “Guidelines for process model libraries using an object-oriented approach.” In *Proceedings of the European Simulation Multiconference, ESM’94*, pp. 349–353. SCS, The Society for Computer Simulation, Barcelona, Spain.

Nilsson, B. (1996): “Experiences of Developing Process Model Libraries in Omola.” In Davis *et al.*, Eds., *International Conference on Intelligent Systems in Process Engineering, ISPE’95*, vol. 92 of *AIChE Symposium Series no. 312*, pp. 388–392. CACHE and AIChE.

Nilsson, B. and J. Eborn (1995): “An object-oriented model database for thermal power plants.” In Breitenecker and Husinsky, Eds., *Eurosim ’95 Simulation Congress*, pp. 747–752. Elsevier.

Patankar, S. V. (1980): *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation, Taylor & Francis Group, New York.

Pfafferott, T. and G. Schmitz (2000): “Numeric simulation of an integrated CO_2 cooling system.” In *Modelica 2000 Workshop Proceedings*, pp. 89–92. Modelica Association, Lund.

Reid, R. C., J. M. Prausnitz, and B. E. Poling (1987): *The Properties of Gases and Liquids.* Mc Graw Hill, Boston, Massachusetts.

Sørlie, J. and J. Eborn (1998): “Parameter optimization results for a family of thermo-physical drum boiler models.” In *Preprints of Reglermöte ’98*, pp. 131–136. Lund Institute of Technology, Sweden.

Stodola, A. (1927): *Steam and Gas Turbines*, sixth edition. McGraw-Hill, New York.

Traupel, W. (1977): *Thermische Turbomaschinen*, third edition. Springer Verlag, Berlin, Germany.

Tummescheit, H. (2000a): “Object-oriented modeling of physical systems, part 11.” *Automatisierungstechnik*, **48:2**. In german.

Tummescheit, H. (2000b): “Object-oriented modeling of physical systems, part 12.” *Automatisierungstechnik*, **48:4**. In german.

Tummescheit, H. and J. Eborn (1998): “Design of a thermo-hydraulic model library in Modelica.” In Zobel and Moeller, Eds., *Proc. of the 12th European Simulation Multiconference, ESM’98*, pp. 132–136. SCS, Manchester, UK.

Wagner, F. J. and M. Z. Poulsen (1999): “C++ toolbox for object oriented modeling and dynamic simulation of physical systems.” *SIMS Conference, Linköping, Sweden.*

Wagner, F. J., M. Z. Poulsen, P. G. Thomsen, and N. Houbak (1998): “Object oriented toolbox for modeling and simulation of dynamical systems.” *SIAM Workshop.*

Wagner, W. and A. Kruse (1998): *Properties of water and steam*. Springer, Berlin.

Paper III

Flow Instabilities in Boiling Two Phase Flow

Jonas Eborn, Hubertus Tummescheit and Karl Johan Åström

Abstract

Tubes with boiling are common elements of many processes. They appear in steam generators and refrigerators and many other systems. The behavior of such systems is complicated because many physical phenomena are involved. It has for example been observed that different types of instabilities can occur. In this paper we will discuss modeling of tubes with boiling. As an application we will discuss an instability phenomenon due to pressure oscillations that has been observed experimentally in many different situations. We will first derive a simple analytical model which is able to capture the oscillations qualitatively. The simple model also gives insight into the mechanisms that generate the oscillations. A more detailed model is then built using a recently developed model base library in Modelica. A comparison between the simple and the detailed model is also presented.

Extended version of

Eborn, J. and K. J. Åström (2000): “Modeling of a boiler pipe with two-phase flow instabilities.” In *Modelica 2000 Workshop Proceedings*, pp. 79–88. Modelica Association, Lund, Sweden.

1. Introduction

Evaporation of fluids flowing through a tube is common in many processes. It is a key element in steam generators, refrigerators and many other systems. The physical phenomena during evaporation is quite complicated. Both the dynamics and the material properties of the fluid are highly non-linear and key quantities like the dry-out point or the amount of superheat are difficult to measure. Many factors contribute to making these processes hard to control. It has for example been observed that different types of flow instabilities/oscillations can occur, see Aldridge and Fowler (1996); Kakaç and Liu (1991); Yadigaroglu (1981).

In this paper we will discuss modeling of tubes with boiling. As an application we will discuss an instability phenomenon due to pressure oscillations that has been observed experimentally in different situations, e. g., in Liu *et al.* (1995). Modeling of such systems is usually done by "brute-force", using CFD code with high discretization. In this paper we take a different approach.

Within the framework of a European collaboration a new modeling language, called Modelica, has been developed. Modelica is based upon the experiences of the members of the Modelica Design Group and is aiming at becoming a standard for equation-based continuous-time and hybrid modeling. As a part of the effort some Modelica base libraries for applications within different domains have been developed, among these a thermo-hydraulic base library, see Tummescheit *et al.* (2000). This library is used here to build a discretized model of a boiling tube.

First a simple, low-order analytical model is derived from first principles. The simple model is able to capture the oscillations that have been observed experimentally. It can also give insight into the mechanisms that generate the oscillations. Then a more complex model is built using the thermo-hydraulic base library in Modelica. Simulations of the complex model shows that it gives results comparable to the simplified model and also close to experimental data. But the complex model also has a richer behavior. Comparisons between models of different complexity and based on different assumptions are presented.

2. A Simple Physical Model

Liquid streaming through a heated tube is heated gradually to boiling temperature, after this point there is a two-phase mixture of liquid and vapor. If sufficient heat is supplied all liquid will evaporate and after the dry-out point there is only vapor which may become super-heated. This is illustrated schematically in Figure 1 which shows the mass fraction of

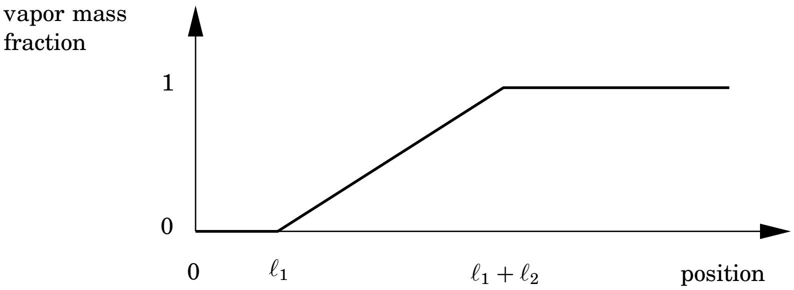


Figure 1 Distribution of vapor mass fraction along a heated tube.

vapor along the tube. The heating zone is from 0 to ℓ_1, the boiling zone from ℓ_1 up to $\ell_1 + \ell_2$ and after this point there is no liquid left in the tube. If the flow rate is too high the liquid does not spend enough time in the tube for all liquid to evaporate and there will only be two zones. If the flow rate is even higher boiling may not even start.

Two phase flows are quite complicated. Here we will use a simplified, homogeneous model. Let P be the heat flow per unit length supplied to the tube, let $\dot{m}$ be the mass flow rate, let h_{in} be the enthalpy of the liquid at the entrance of the tube, h_l the liquid enthalpy at boiling temperature and $h_c = h_v - h_l$ the difference between the enthalpy of the vapor and the liquid. Moreover let L be the length of the tube and ℓ_1 and ℓ_2 be the length of the heating and boiling zones. If we assume stationary conditions, a global energy balance gives [1]

$$\begin{aligned} \dot{m}(h_l - h_{in}) &= P\ell_1 \\ \dot{m}h_c &= P\ell_2 \end{aligned} \tag{1}$$

Complete boiling occurs if the mass flow rate is sufficiently low, i. e., $\dot{m} < \dot{m}_c$, where $\dot{m}_c$ is defined as the critical flow rate. This occurs exactly when $\ell_1 + \ell_2 = L$, which gives us the critical flow rate

$$\dot{m}_c = \frac{PL}{h_v - h_{in}} \tag{2}$$

Let v be the velocity, ρ the density of the liquid-vapor mixture, and $\dot{m}$ the mass flow rate. Assuming that we have a common $\dot{m}$ through the

[1] To be formally correct, the energy balance should use internal energy, u, instead of h. In the liquid part, $(h_l - h_{in})$, the error is negligible, about 0.1%. However, on the vapor side the error can be around 5%. For more accurate calculations, we should instead use $h_c = h_v - h_l - p_v v_v$ to correct for the pressure volume work in the evaporation. As a simplifying assumption we have chosen to neglect this.

entire tube, i. e., no pressure waves, the frictional pressure drop can be expressed as

$$\Delta p = \frac{k}{2}\overline{\rho v^2} = \frac{k}{2A^2}\frac{\dot{m}^2}{\overline{\rho}} \tag{3}$$

where the bar denotes average values over the entire tube. To determine the pressure drop we thus have to calculate the average $1/\overline{\rho}$. We will consider three separate cases.

Complete Boiling: In this case all phases are present. In Bell and Åström (2000) it was shown that in steady state the mass ratio of the vapor in a heated tube is piecewise linear. If we assume that this profile is a good approximation also in the transient stage we can assume that the volumity, $\nu = 1/\rho$, is an affine function in the boiling zone, i.e.

$$\nu = \frac{\xi}{\ell_2}\nu_v + \frac{\ell_2 - \xi}{\ell_2}\nu_l$$

where $0 \leq \xi \leq \ell_2$ and the origin is at the start of boiling. The average volumity is then

$$\begin{aligned}\frac{1}{\overline{\rho}} = \overline{\nu} &= \frac{1}{L}\Big(\int_0^{\ell_1}\nu_l d\xi + \int_0^{\ell_2}(\nu_l + \xi\frac{\nu_v - \nu_l}{\ell_2})d\xi + (L - \ell_1 - \ell_2)\nu_v\Big) \\ &= \frac{\ell_1}{L}\frac{1}{\rho_l} + \frac{\ell_2}{2L}\Big(\frac{1}{\rho_v} + \frac{1}{\rho_l}\Big) + \frac{L - \ell_1 - \ell_2}{L}\frac{1}{\rho_v}\end{aligned} \tag{4}$$

It follows from Equation (1) and Equation (2) that

$$\frac{\ell_1}{L} = \frac{h_l - h_{in}}{h_v - h_{in}}\frac{\dot{m}}{\dot{m}_c} \qquad \frac{\ell_2}{L} = \frac{h_v - h_l}{h_v - h_{in}}\frac{\dot{m}}{\dot{m}_c}$$

Introducing $x = m/m_c$ into (4) we find that

$$\frac{\rho_l}{\overline{\rho}} = (1 - x)\frac{\rho_l}{\rho_v} + x\frac{h_l - h_{in}}{h_v - h_{in}} + \frac{x}{2}\frac{h_v - h_l}{h_v - h_{in}}\frac{\rho_l + \rho_v}{\rho_v}$$

Partial Boiling: In this case there is only a heating zone and a boiling zone. The flow at the exit of the tube consists of a mixture of both vapor and liquid. We have $\ell_1 < L = \ell_1 + \ell_2$ and the average volumity is given by

$$\begin{aligned}\frac{1}{\overline{\rho}} = \overline{\nu} &= \frac{1}{L}\Big(\int_0^{\ell_1}\nu_l d\xi + \int_0^{\ell_2}(\nu_l + a_r\xi\frac{\nu_v - \nu_l}{\ell_2})d\xi\Big) \\ &= \frac{\ell_1}{L}\frac{1}{\rho_l} + \frac{\ell_2}{2L\rho_l}\Big(2 + a_r(\frac{\rho_l}{\rho_v} - 1)\Big)\end{aligned} \tag{5}$$

where a_r is the mass fraction of vapor at the tube outlet. Neglecting the energy increase in the pure vapor phase a global energy balance gives

$$\dot{m}a_r(h_v - h_l) = P\ell_2 = P(L - \ell_1) = \dot{m}_c(h_v - h_{in}) - \dot{m}(h_l - h_{in})$$

Combining this with Equation (2) we find

$$\begin{aligned} a_r &= \frac{\dot{m}_c(h_v - h_{in}) - \dot{m}(h_l - h_{in})}{\dot{m}(h_v - h_l)} \\ \frac{\ell_2}{L} &= \frac{\dot{m}}{\dot{m}_c}\frac{h_v - h_l}{h_v - h_{in}}a_r = 1 - \frac{\dot{m}}{\dot{m}_c}\frac{h_l - h_{in}}{h_v - h_{in}} \end{aligned}$$

Inserting this into (5) we get

$$\frac{\rho_l}{\overline{\rho}} = x\frac{h_l - h_{in}}{h_v - h_{in}} + \frac{1}{2}\Big(1 - x\frac{h_l - h_{in}}{h_v - h_{in}}\Big)\Big(2 + \frac{h_v - h_{in} - x(h_l - h_{in})}{x(h_v - h_l)}\frac{\rho_l - \rho_v}{\rho_v}\Big)$$

No Boiling: In the case when there is no boiling we have

$$\frac{\rho_l}{\overline{\rho}} = \frac{\rho_l}{\rho_l} = 1$$

Summary: For the three different cases we find that the pressure drop is given by

$$\Delta p = \frac{k\dot{m}^2}{2A^2\overline{\rho}} = \frac{k\dot{m}_c^2}{2A^2\rho_l}f\Big(\frac{\dot{m}}{\dot{m}_c}\Big) \tag{6}$$

where the function $f = x^2\rho_l/\overline{\rho}$ is given by

$$f(x) = \begin{cases} x^2a_3 + x^3\big(a_1 + \frac{a_2}{2}(a_3 + 1) - a_3\big) & \text{for } 0 \le x < 1 \\ x^2\Big(xa_1 + \frac{1 - a_1x}{2}\big(2 + \frac{1 - a_1x}{a_2x}(a_3 - 1)\big)\Big) & \text{for } 1 \le x < \frac{1}{a_1} \\ x^2 & \text{for } x \ge \frac{1}{a_1} \end{cases} \tag{7}$$

where the coefficients a_i are given by

$$a_1 = \frac{h_l - h_{in}}{h_v - h_{in}} \qquad a_2 = \frac{h_v - h_l}{h_v - h_{in}} \qquad a_3 = \frac{\rho_l}{\rho_v} \tag{8}$$

Figure 2 shows the normalized pressure drop as a function of normalized mass flow rate. Notice that the curve has a negative slope if the density ratio ρ_ℓ/ρ_v is sufficiently large. When this occurs the function f also has nontrivial extrema. The plot also shows the normalized pressure drop for pure liquid and pure vapor ($\rho_\ell/\rho_v = 100$). For low mass flow rate the pressure drop for the mixture is close to the curve for pure vapor and for high mass flow rates the pressure drop approaches the curve for pure liquid.

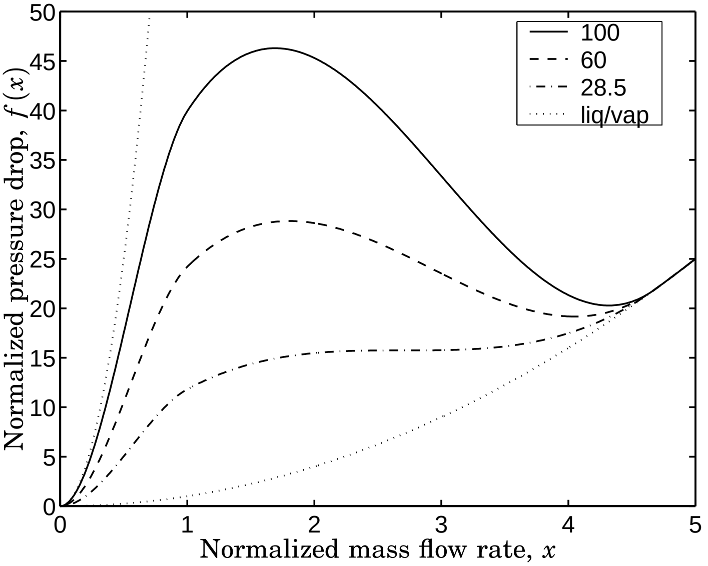


Figure 2 The normalized pressure drop as a function of normalized mass flow rate for different density ratios, ρ_l/ρ_v. Dotted lines are the limiting cases, pure liquid and pure vapor flow. Enthalpy values are taken from Table 1.

No subcooling

The function f simplifies considerably if the liquid entering the tube is close to boiling temperature, i. e., with very little sub-cooling. Then $a_1 = 0$ and $a_2 = 1$, and by substituting a_3 in (7) the function f becomes

$$f(x) = \begin{cases} x^2 \dfrac{\rho_l}{\rho_v} - \dfrac{x^3}{2}\Big(\dfrac{\rho_l}{\rho_v} - 1\Big) & \text{for } 0 \leq x < 1 \\ x^2 + \dfrac{x}{2}\Big(\dfrac{\rho_l}{\rho_v} - 1\Big) & \text{for } 1 \leq x \end{cases} \tag{9}$$

Notice that the shape of the function f now only depends on the ratio of the densities of the liquid and the vapor phase, ρ_l/ρ_v. Differentiation of Equation(9) gives

$$f'(x) = \begin{cases} 2x \dfrac{\rho_l}{\rho_v} - \dfrac{3x^2}{2}\Big(\dfrac{\rho_l}{\rho_v} - 1\Big) & \text{for } 0 \leq x < 1 \\ 2x + \dfrac{1}{2}\Big(\dfrac{\rho_l}{\rho_v} - 1\Big) & \text{for } 1 \leq x \end{cases} \tag{10}$$

Since the coefficients are positive ($\rho_l/\rho_v \geq 1$) the derivative expressions are always positive. Thus, the pressure drop is a strictly increasing function and *there can be no pressure drop oscillations without subcooling*.

Conditions for local extrema

As can be seen in Figure 2 the pressure drop function (7) can have negative slope for certain values of ρ_l/ρ_v. The local maximum and minimum will, for reasonable values of a_i, always occur in the partial boiling region, $1 \leq x < 1/a_1$. By differentiating the expression for f in this region and setting it equal to zero we obtain after some simplification

$$f'(x) = 3x^2\Big(\frac{a_1^2(a_3-1)}{2a_2}\Big) + 2x\Big(1 - \frac{a_1(a_3-1)}{a_2}\Big) + \frac{a_3-1}{2a_2} = 0 \qquad (11)$$

$$\Leftrightarrow 3(xa_1)^2 + 4(xa_1)\Big(\frac{a_2}{a_1(a_3-1)} - 1\Big) + 1 = 0 \qquad (12)$$

Solving this equation we find that the local extrema occur at

$$x_{1,2} = \frac{2}{3a_1}\Big(1 - \frac{a_2}{a_1(a_3-1)}\Big) \pm \frac{1}{a_1}\sqrt{\frac{4}{9}\Big(1 - \frac{a_2}{a_1(a_3-1)}\Big)^2 - \frac{1}{3}} \qquad (13)$$

Examining the second derivative shows that it is negative for small x and that it becomes positive for x-values larger than $x_0 \approx 2/(3a_1)$, indicating a maximum for the smaller x in (13) followed by a minimum. The condition for having two separate extrema is

$$1 \leq \frac{4}{3}\Big(1 - \frac{a_2}{a_1(a_3-1)}\Big)^2$$

which by solving for the density ratio can be written as

$$\frac{\rho_l}{\rho_v} = a_3 \geq a_3^c = 1 + \frac{a_2}{a_1}(4 + 2\sqrt{3}) \qquad (14)$$

The solution to this inequality shows that if the density ratio, ρ_l/ρ_v, is sufficiently large the conditions for local extrema are fulfilled and pressure-drop oscillations can occur. The critical density ratio (14) depends on a_1, a_2 and thus on the amount of sub-cooling. The conditions in Figure 2 are given in Table 1, which gives the critical density ratio $\rho_l/\rho_v = 28.5$.

Summarizing we find that if ρ_l/ρ_v fulfills condition (14) then the function f has two extrema given by (13) and the pressure drop function f has negative slope between the extrema. The extrema will coincide if $\rho_l/\rho_v = a_3^c$. The distance between the extrema $x_2 - x_1$ increases with increasing density ratio ρ_l/ρ_v.

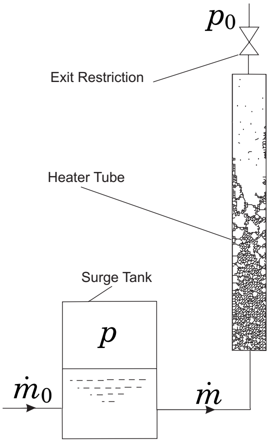


Figure 3 Schematic of a boiling channel.

3. Analysis of Oscillations

Having obtained a model for the pressure drop we will now investigate some interesting dynamical phenomena. Figure 3 shows a schematic diagram of a boiling channel with a surge tank. This corresponds to the experimental configuration used in several experiments, see Liu *et al.* (1995).

Let $\dot{m}_0$ and $\dot{m}$ denote the mass flow rate in and out of the surge tank respectively. Let p be the pressure in the surge tank and p_0 the external pressure. The system can be described by mass and momentum balances. The variables p and $\dot{m}$ are chosen as states variables.

A momentum balance for the heater tube gives

$$\frac{d\dot{m}}{dt} = \frac{A}{L}(p - p_0 - \Delta p) = \frac{A}{L}(p - p_0 - \frac{k\dot{m}_c^2}{2A^2\rho_l}f(\frac{\dot{m}}{\dot{m}_c})) \tag{15}$$

where A is the cross-section of the tube and L is the length of the tube.

A mass balance for the surge tank gives the differential equation for liquid volume, V, as

$$\rho_l \frac{dV}{dt} = \dot{m}_0 - \dot{m} \tag{16}$$

Let V_t be the total volume of the surge tank and let V_0 be the volume of gas in the surge tank in the normal state, when the pressure is p_0. The pressure in the tank is given by the ideal gas law

$$p(V_t - V) = p_0 V_0$$

To obtain a differential equation for p this expression is differentiated, hence

$$\frac{dp}{dt}(V_t - V) = p\frac{dV}{dt}$$

Solving for the derivative of p and eliminating $V_t - V$ we get

$$\frac{dp}{dt} = \frac{p}{V_t - V}\frac{dV}{dt} = \frac{p^2}{\rho_l p_0 V_0}(\dot{m}_0 - \dot{m})$$

The system is thus described by the second order differential equation system

$$\begin{aligned}\frac{d\dot{m}}{dt} &= \frac{A}{L}\Big(p - p_0 - \frac{k\dot{m}_c^2}{2A^2\rho_l} f(\frac{\dot{m}}{\dot{m}_c})\Big)\\ \frac{dp}{dt} &= \frac{p^2}{\rho_l p_0 V_0}(\dot{m}_0 - \dot{m})\end{aligned} \tag{17}$$

Equilibria

The system Equation (17) has an equilibrium which is given by

$$\begin{aligned}\dot{m} &= \dot{m}_0\\ p &= p_0 + \frac{k\dot{m}_c^2}{2A^2\rho_l} f(\frac{\dot{m}_0}{\dot{m}_c})\end{aligned} \tag{18}$$

Normalization

Introduce the time constants

$$T_{ma} = \frac{\rho_l V_0}{\dot{m}_0} \qquad \text{and} \qquad T_{mo} = \frac{\dot{m}_0 L}{A p_0}$$

which are associated with the mass balance of the surge tank and the momentum balance of the heating tube respectively. Notice that $\rho_l V_0$ is the mass of the liquid in the volume V_0 and $\dot{m}_0$ is the mass flow rate. Similarly, note that $\dot{m}_0 L$ is the momentum of the fluid in the tube and $A p_0$ is the force acting on the fluid.

Furthermore, introducing the scaled time τ defined by

$$\tau = \sqrt{\frac{1}{T_{ma}T_{mo}}}t = \sqrt{\frac{A p_0}{\rho_l V_0 L}}t$$

the equation system (17) becomes

$$\begin{aligned}\frac{dx}{d\tau} &= \alpha(y - 1 - \beta f(\gamma x))\\ \frac{dy}{d\tau} &= \frac{1}{\alpha}(1 - x)y^2\end{aligned} \tag{19}$$

where

$$\alpha = \sqrt{\frac{\rho_l V_0 A p_0}{L \dot{m}_0^2}} = \sqrt{\frac{T_{ma}}{T_{mo}}}, \qquad \beta = \frac{k \dot{m}_c^2}{2A^2 \rho_l p_0} = \frac{\Delta p_c}{p_0}, \qquad \gamma = \frac{\dot{m}_0}{\dot{m}_c}$$

and Δp_c is the stationary pressure drop when $\dot{m} = \dot{m}_0 = \dot{m}_c$.

If the normalized model (19) is used the system is thus characterized by four parameters only:

- The ratio of the densities ρ_l / ρ_v.
- The ratio of the time constants associated with the mass balance (T_{ma}) and the momentum balance (T_{ma}).
- The ratio $\Delta p_c / p_0$
- The ratio $\dot{m}_0 / \dot{m}_c$

Linearization

The equilibrium values of the normalized variables are

$$\begin{aligned} x &= 1 \\ y &= y_0 = 1 + \beta f(\gamma) \end{aligned}$$

If we choose $u = \Delta m_0 / m_0$ as an input and linearize the system around the equilibrium we find

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -\alpha\beta\gamma f'(\gamma) & \alpha \\ -y_0^2/\alpha & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ y_0^2/\alpha \end{pmatrix} u \qquad (20)$$

Since the coefficients α, β and γ are positive it follows that the equilibrium is stable if $f'(\gamma)$ is positive and that it is unstable if $f'(\gamma)$ is negative.

Simulation

The simple model (17) can easily be coded in Modelica to examine the behavior of the model. The pressure drop function, f, is given as a Modelica function with four arguments, as seen in the code example below:

```
model evap "Evaporating tube w. density-varying pressure drop"
  parameter SIunits.Power P(start=800);
    ⋮
  SIunits.Pressure dp;
  SIunits.Pressure p(start=3e5);
  SIunits.MassFlow m(start=0.024);
```

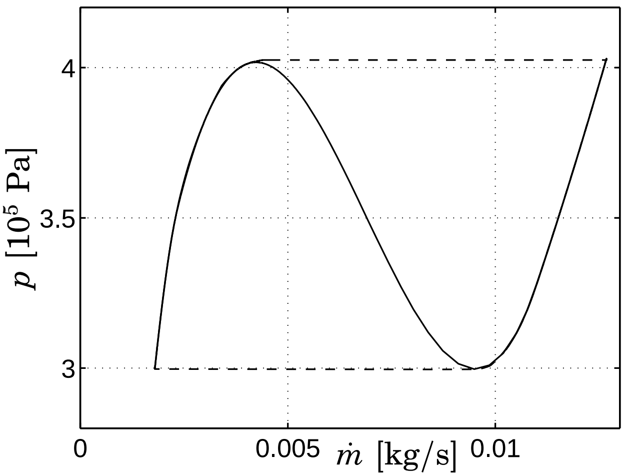


Figure 4 Simulated limit cycle of pressure drop oscillations, — pressure drop characteristic, – – pressure in the surge tank.

```
equation
  dp = k*mc*mc/(2*A*A*rl)*f(x=m/mc, a3=rl/rv, a1=a1, a2=a2);
  der(p) = p*p*(m0 - m)/p0/V0/rl;
  der(m) = A/L*(p - pe - dp);
end evap;
```

Parameters for the simple model, taken from Liu *et al.* (1995), are given in Table 1. With these parameters the model gives a limit cycle in pressure and mass flow which agrees well with the measurements in Liu *et al.* (1995). A phase plot of the limit cycle is shown in Figure 4. The pressure drop characteristic is drawn with a full line and the pressure in the surge tank is dashed. The period of the oscillation depends critically on the volume of the surge tank, V_0, which is not stated in the paper Liu *et al.* (1995).

4. More Complex Modelica Models

Modeling in Modelica can be easy. With the available libraries, simply select and connect modules in a graphical editor. Modelica libraries are open which means that the models can be modified. Here we have used the Modelica base library for thermo-hydraulic models, ThermoFlow, described in

Name	Value	Unit	Quantity
P	800	W	heat input
L	0.605	m	length
d	7.5	mm	diameter
ρ_v	22.5	kg/m^3	vapor density
ρ_l	1359	kg/m^3	liquid density
h_v	426	kJ/kg	vapor enthalpy
h_l	264	kJ/kg	liquid enthalpy
h_{in}	220	kJ/kg	inflow enthalpy
p_e	10^5	Pa	exit pressure
p_0	10^5	Pa	normal pressure
V_0	0.7	l	normal volume
$\dot{m}_0$	7.31	g/s	mass flow rate
k	10^4	–	friction factor

Table 1 Parameter values for the simplified model. Property data (h, ρ) are mean values for R11 from REFPROP, see McLinden *et al.* (1998). The friction pressure drop also includes exit and entry losses, which makes the friction factor, k, quite large.

Tummescheit *et al.* (2000); Tummescheit (2000). The basic components in this library are lumped and discretized control volumes containing the balance equations for mass, energy and momentum.

To verify the simplified model we compare it with a more complex, discretized model. Using the standard components in ThermoFlow, a system model shown in Figure 5 is created with a surge tank and a discretized pipe. A nice feature of Modelica and the ThermoFlow library is that the degree of discretization can be expressed by a parameter, n, which gives the number of sections. In the results shown here we have used $n = 10$, which gave reasonable results.

A complex model can exhibit a much more complicated behavior than the simplified one. One reason for this is that a discretized model also captures other physical phenomena, like the density-wave oscillations, see Yadigaroglu (1981).

Different modeling assumptions

The fact that it is easy to build models of different complexity in Modelica can be used to develop models based on different assumptions. This can help getting insight into the behavior of the system.

The simplest model has only two states, one pressure and one mass flow. A fully discretized model will have $3n+1$ states, since we discretize 3 balance equations in n sections and include one state in the surge tank. It would also be interesting to test another assumption; to discretize different balance equations differently.

A key issue in physical modeling is to determine how accurate different storage quantities should be represented. With the choice of states used in ThermoFlow, pressure represents mass storage and storage of thermal energy is represented by enthalpy. It is only convenience that suggests having the same resolution for the storage of mass and energy. In the following section a model using two different resolutions is developed.

We have then three different model structures for comparison, using different assumptions:

$\mathcal{M}_1$: The simplified model, with only 2 states.

$\mathcal{M}_2$: The one-flow model, with discretized energy balance and lumped mass balance, developed in the following section.

$\mathcal{M}_3$: A fully discretized model, with $3n+1$ states.

One flow, multi-temperature model

We wish to obtain a model similar to the simplified model, but using a discretized energy balance to obtain the pressure drop in the tube from the real density variation in the tube, and not the simplified function derived in Section 2. To do this we use a lumped mass (pressure) balance and a discretized thermal (enthalpy) balance. In this way there is only one

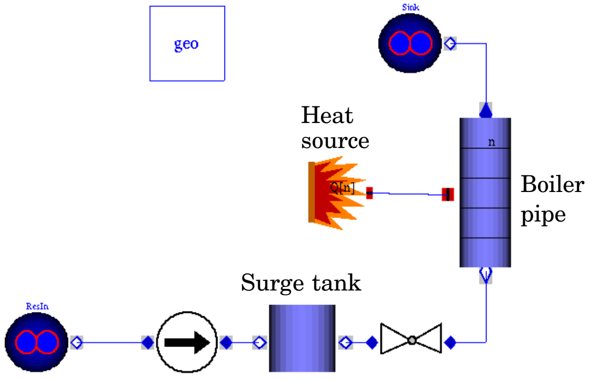


Figure 5 Model diagram of the system.

flow through the pipe instead of n flows between the discretized volumes. Thus we can only obtain the type of pressure-drop oscillations seen with the simplified model. This model is not available as a component in the library, but can be obtained by changing some of the basic components. Figure 6 shows a schematic of the pipe model, the different base classes that together form the model and how they are discretized in the one-flow model. Below we also give the code of the central parts of the pipe model; BalanceTwoPort, ThermalModel, MediumModel and FlowModel. The essential code with the changes to the classes in ThermoFlow is listed here.

For the model to be correct, it is important to keep track of the mean thermal state, h_{mean}, used in the ThermalModel below. It is also important to distribute the flow difference between inlet and outlet over the discretized energy balances, which can be seen in the BalanceTwoPort. Otherwise flow changes are concentrated in one section of the pipe, influencing the thermal state in that section too much.

```
partial model BalanceTwoPortSingleSpecial
  ...
equation
  ...
  edot[1] = a.q_conv;
  for i in 2:n loop    // Interpolated mass flow used in edot
    edot[i] = if mdot[1]-dM[1]*(i-1) > 0
      then (mdot[1]-dM[1]*(i-1))*h[i-1]
      else (mdot[1]-dM[1]*(i-1))*h[i];
  end for;
  edot[n+1] = if mdot[2]>0 then mdot[2]*h[n] else mdot[2]*b.h;
  dM = (mdot[1] - mdot[2])*ones(n)/n;
  for i in 1:n loop
```

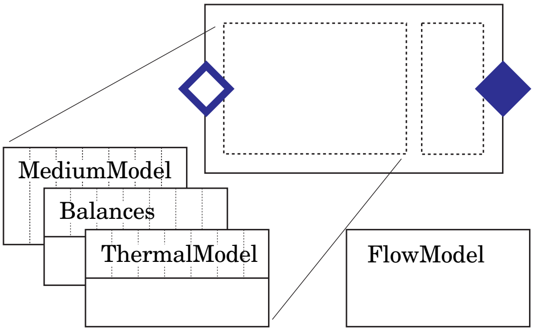


Figure 6 Basic classes in a pipe model with different discretization for mass and energy balances.

```
    dU[i]) = edot[i] - edot[i+1] - p[1]*der(V[i]) + Q_s[i];
  end for;
end BalanceTwoPortSingleSpecial;

partial model ThermalModelSpecial
  replaceable model Medium = StateVariablesSpecial;
  extends Medium;
equation
  ...
  for i in 1:n loop      // thermal state equations
    km[i]*der(h[i]) = kh[1, i]*dM[i] + kh[2, i]*dU[i];
  end for;
  km_mean*der(p[1]) = kp[1]*sum(dM) + kp[2]*sum(dU);
  // Mean value of enthalpies, used for mean thermal state
  h_mean = h[1:n]*d[1:n]/sum(d[1:n]);
end ThermalModelSpecial;
```

The medium model used here is almost the standard one; only the extra calculation of the mean thermal state has to be included. This is done in the last element of the properties record, `pro[n+1]`. The flow model is also very close to an ordinary lumped flow model, the only difference is the summation of the pressure losses.

```
model WaterSteamSpecial
  extends StateVariablesSpecial;
  Integer phase[n + 1];
equation
  // Region check with events, only covers regions 1, 2 and 4
  for i in 1:n loop
    phase[i] = if ((h[i] < SteamIF97.hlofp(p[1])) or
             (h[i] > SteamIF97.hvofp(p[1]))) then 1 else 2;
    pro[i] = Water.water_ph(p[1],h[i],phase[i]);
  end for;
  phase[n+1] = if ((h_mean < SteamIF97.hlofp(p[1])) or
             (h_mean > SteamIF97.hvofp(p[1]))) then 1 else 2;
  pro[n+1] = Water.water_ph(p[1],h_mean,phase[n+1]);
end WaterSteamSpecial;

model FlowModelTwoPortSingleSpecialDyn
  ...
equation
  G_norm[2] = if mdot[2] > 0
    then mdot[2]*mdot[2]/d_mean/A
    else -mdot[2]*mdot[2]/ddown/A;
  dG = G_norm[1]-G_norm[2] + dGdown;
  // This is the momentum balance equation
  L*der(mdot[2]) = dG + (p[1] - pdown)*A
```

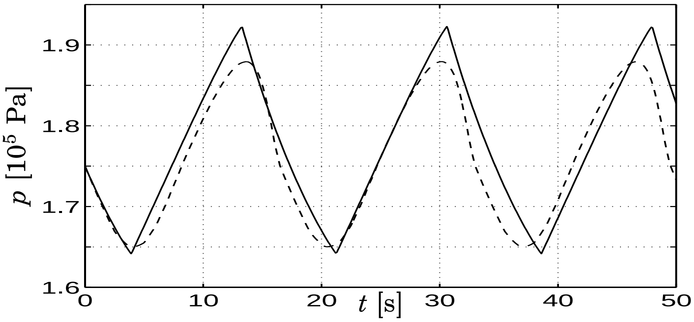


Figure 7 Pressure drop oscillations with simplified (—) and complex model (– –). Note that the medium here is water, this plot should not be compared with Figure 4.

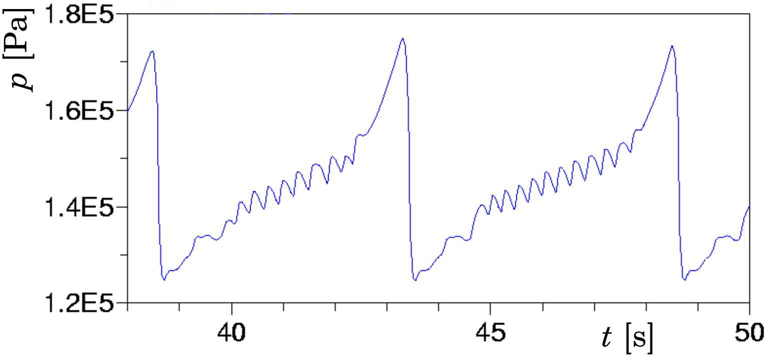


Figure 8 High-frequency oscillations with fully discretized model. Parameters are slightly different compared to the previous case.

```
                    - sum(Ploss)/n*L*Dhyd*Pi;
end FlowModelTwoPortSingleSpecialDyn;
```

Simulating model $\mathcal{M}_2$ gives results similar to the simplified model $\mathcal{M}_1$, see Figure 7. This verifies that the derivation of the pressure loss in the simplified model is correct. The simulation results are also qualitatively very close to the experimental results in Liu *et al.* (1995).

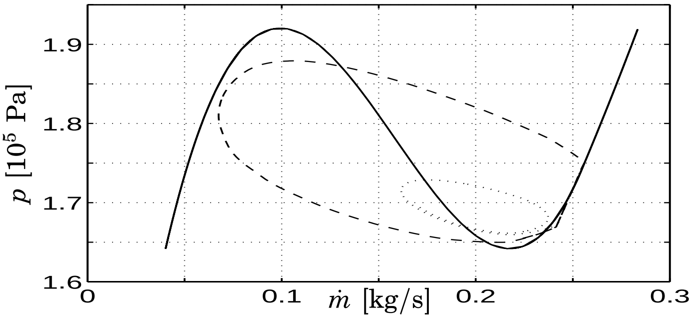


Figure 9 Simulation results of the complex model, characteristic obtained from the simplified model. Two limit cycles for different mean flows, $\dot{m} = 0.15$ kg/s (– –) and 0.2 kg/s (- - -).

Fully discretized model

Model $\mathcal{M}_3$ uses the standard model of a discretized pipe with n mass balances (and flows). Using $\mathcal{M}_3$ to study the problem with pressure oscillations gives results slightly different due to discretization effects. Each time one section of the pipe goes from liquid to two-phase it generates a small pressure shock wave in the system. When there are pressure oscillations the phase of the sections is constantly changing and thus high-frequency shock waves are generated, superimposed on the slower pressure oscillations, see Figure 8. The number of high-frequency peaks is partly due to the number of discrete sections in the model. These peaks could also emanate from high-frequency density-wave oscillations, Yadigaroglu (1981). But since they are driven by the discretization artifacts we cannot tell. One way around this problem would be to use a so-called moving boundary model, Heusser (1996), but this has not been done here.

5. Comparisons

The pressure drop oscillations observed using the simplified model $\mathcal{M}_1$ are similar in period and amplitude to the oscillations obtained with the one-flow model $\mathcal{M}_2$. Note that the shape of the pressure drop curve, and thus the properties of the oscillations, depend very much on the average media properties used, ρ_l, ρ_v, h_l, h_v and h_{in}. In the complex models the properties change with the pressure and mass flow into the system, which causes differences in limit cycle period and amplitude. In Figure 9 the simplified pressure drop characteristic is plotted together with limit cycles obtained with model $\mathcal{M}_2$. The amplitude and damping of the oscillations vary with the mean flow. This is caused by the energy dynamics which produce a lag in the density changes, unlike the immediate response of the pressure drop function in the simplified model. Model $\mathcal{M}_1$ uses constant, average properties and thus gives the same limit cycle amplitude for all mass flows within the unstable region.

6. Conclusions

A simplified model of a boiler tube has been derived, assuming equilibrium conditions and a linear quality profile also during transients. The model gives a closed expression for the pressure drop which depends on mass flow through the tube and the density ratio of vapor and liquid. The simplified model gives insight into how a known instability phenomenon, pressure-drop oscillations, arises. The simplified model also gives results close to measurements in Liu *et al.* (1995).

The simplified model has also been compared to two different discretized models developed in Modelica. A one-flow model with lumped mass balance and discretized energy balance and a fully discretized model. The one-flow model is shown to give more realistic oscillations than the simplified model. The fully discretized model, however, gives high frequency oscillations due to discretization effects and is not reliable for studies of pressure-drop oscillations.

The two discretized models was built using a thermo-hydraulic base library in Modelica, ThermoFlow. The example shows how models for studying a complicated phenomenon can be built from model library components, and how the library components can be adapted. The results also illustrate how the flexibility of Modelica makes it possible to explore a wide range of models, with less effort.

References

Aldridge, C. J. and A. C. Fowler (1996): "Stability and instability in evaporating two-phase flows." *Surveys on Mathematics for Industry*, **6**, pp. 75–107.

Bell, R. D. and K. J. Åström (2000): "Drum-boiler dynamics." *Automatica*, **36:3**, pp. 363–378.

Heusser, P. A. (1996): *Modelling and Simulation of Boiling Channels with a General Front Tracking Approach*. SCS.

Kakaç, S. and H. T. Liu (1991): "Two-phase flow dynamic instabilities in boiling systems." In Chen *et al.*, Eds., *Proc. 2nd Int. Symp. on Multi-Phase Flow and Heat Transfer*, vol. 1, pp. 403–444.

Liu, H. T., H. Koçak, and S. Kakaç (1995): "Dynamical analysis of pressure-drop type oscillations with a planar model." *Int.J. Multiphase Flow*, **21:5**, pp. 851–859.

Tummescheit, H. (2000): "Object-oriented modeling of physical systems, part 11." *Automatisierungstechnik*, **48:2**. In german.

Tummescheit, H., J. Eborn, and F. Wagner (2000): "Development of a Modelica base library for modeling of thermo-hydraulic systems." In *Modelica 2000 Workshop Proceedings.* Modelica Association, Lund.

Yadigaroglu, G. (1981): "Two-phase flow instabilities and propagation phenomena." In Delhaye *et al.*, Eds., *Thermal-hydraulics of Two-phase Systems for Industrial Design and Nuclear Engineering*, chapter 17. Hemisphere McGraw-Hill, New York.

Appendix IIIa

Appendix to

Flow Instabilities in Boiling Two Phase Flow

Here the Modelica code for the models in Paper III is listed. All graphical annotations have been taken out. First we give the complete code for the simplified model, including the definition of the pressure drop function (7). The next section lists the code that was adapted to run the one-flow model. This means that the code for the PipeModel is complete, except for the steam-table functions, and also some of the simpler components in the system model that has been left out. The actual system model is at the end of the appendix. The code for the fully discretized model $\mathcal{M}_3$ is in major parts the same as for model $\mathcal{M}_2$ and has not been included here.

Modelica code for the simplified model, $\mathcal{M}_1$

```
package Pdrop

function f
  input Real x;
  input Real a3=100;
  input Real a1=0.000001;
  input Real a2=1.0;
  output Real dp;
algorithm
  if x < 1 then
    dp := x*x*(a3 + x*(a1 + a2*(a3 + 1)/2 - a3));
  else
    if x < 1/a1 then
      dp := x*x*(x*a1 + (1 - a1*x)*(2 + (1 - a1*x)*(a3 - 1)/x/a2)/2);
    else
      dp := x*x;
    end if;
  end if;
end f;
```

```
model evap "Evaporating tube with density-varying pressure drop"
  parameter SIunits.Power P(start=800);
  parameter SIunits.Length L(start=0.605);
  parameter SIunits.Area A=Modelica.Constants.pi*0.00375^2;
  parameter SIunits.Density rv(start=22.5);
  parameter SIunits.Density rl(start=1359);
  parameter SIunits.SpecificEnthalpy hv(start=426e3);
  parameter SIunits.SpecificEnthalpy hl(start=264e3);
  parameter SIunits.SpecificEnthalpy hin(start=220e3);
  parameter SIunits.Pressure p0(start=1e5);
  parameter SIunits.Pressure pe(start=1e5);
  parameter SIunits.Volume V0(start=0.0005);
  parameter SIunits.MassFlowRate m0(start=0.00731);
  parameter SIunits.MassFlowRate mc=P*L/(hv - hin);
  parameter Real k(start=10000);
  parameter Real a1=(hl - hin)/(hv - hin);
  parameter Real a2=(hv - hl)/(hv - hin);
  SIunits.Pressure dp;
  SIunits.Pressure p(start=3e5);
  SIunits.MassFlowRate m(start=0.024);
equation
  dp = k*mc*mc/(2*A*A*rl)*f(x=m/mc, a3=rl/rv, a1=a1, a2=a2);
  der(p) = p*p*(m0 - m)/p0/V0/rl;
  der(m) = A/L*(p - pe - dp);
end evap;

end Pdrop;
```

Modelica code for the one-flow model, $\mathcal{M}_2$

```
package SIunits = Modelica.SIunits;
package CommonRec = Modelica.ThermoFlow.BaseClasses.CommonRecords;
package Interfaces = Modelica.ThermoFlow.Interfaces;
package Balances =
  Modelica.ThermoFlow.BaseClasses.Balances.SingleDynamic;
package Water = Modelica.ThermoFlow.BaseClasses.MediumModels.Water;
package SteamIF97 =
  Modelica.ThermoFlow.BaseClasses.MediumModels.SteamIF97;

package SpecialPipe
  constant Real Pi = Modelica.Constants.pi;

  record ThermoBaseVarsSpec
    parameter Integer n(min=1) = 1 "discretization number";
    SIunits.Pressure p[1];
    SIunits.Temperature T[n], T_mean;
    SIunits.Density d[n], d_mean;
    SIunits.SpecificEnthalpy h[n], h_mean;
    SIunits.Mass M[n](start=ones(n),fixed=false) "Total mass";
    SIunits.Energy U[n](start=ones(n),fixed=false) "Inner energy";
  SIunits.Volume V[n] "Volume";
  SIunits.MassFlowRate dM[n] "Change in total mass";
  SIunits.Power dU[n] "Change in inner energy";
end ThermoBaseVarsSpec;

record ThermoProperties
  "Thermodynamic base property data for all state models"
  SIunits.Temp_K T "temperature";
  SIunits.Density d "density";
  SIunits.Pressure p "pressure";
  SIunits.SpecificEnthalpy h "enthalpy";
  SIunits.SpecificEnergy u "inner energy";
  SIunits.SpecificEntropy s "entropy";
  SIunits.SpecificHeatCapacity cp "heat capacity at const p";
  SIunits.SpecificHeatCapacity cv "heat capacity at const v";
  SIunits.SpecificHeatCapacity R "gas constant";
  SIunits.RatioOfSpecificHeatCapacities kappa "ratio of cp/cv";
  SIunits.DerDensityByEnthalpy ddhp;
  SIunits.DerDensityByPressure ddph;
  SIunits.DerDensityByTemperature ddTp;
  SIunits.DerDensityByPressure ddpT;
  SIunits.DerEnergyByPressure dupT;
  SIunits.DerEnergyByDensity dudT;
  Real duTp "derivative of inner energy by temp at const p";
  SIunits.Velocity a "speed of sound";
end ThermoProperties;

partial model StateVariables_p1hn
  extends ThermoBaseVarsSpec(p(fixed=true), h(fixed=true));
  ThermoProperties_ph pro[n+1];
equation
  for i in 1:n loop
    d[i] = pro[i].d;
    T[i] = pro[i].T;
  end for;
  d_mean = pro[n + 1].d;
  T_mean = pro[n + 1].T;
end StateVariables_p1hn;

record BalanceSetSingleSpecial
  extends ThermoBaseVarsSpec;
  SIunits.Power Q_s[n] "Heat source term";
  SIunits.MassFlowRate mdot[2];
  SIunits.Power edot[n + 1];
  SIunits.MomentumFlux G_norm[2];
  SIunits.MomentumFlux dG;
end BalanceSetSingleSpecial;

partial model BalanceTwoPortSingleSpecial
  extends Interfaces.SingleDynamic.TwoPortAB;
  extends BalanceSetSingleSpecial;
```

```
    extends CommonRec.PressureLossDistributed;
    Interfaces.HeatTransfer.HeatFlowD q(n=n);
    extends CommonRec.ConnectingVariablesSingleDynamic;
  equation
    // Connecting variables needed in flow model:
    ddown = b.d;
    pdown = b.p;
    dGdown = b.dG;
    // Pass state information upstream:
    p[1] = a.p;
    d[1] = a.d;
    h[1] = a.h;
    T[1] = a.T;
    pro[1].s = a.s;
    pro[1].kappa = a.kappa;
    Q_s = q.q;
    T[1:n] = q.T;
    // Flow information,
    // note different sign convention inside distr. for G_n and mdot
    G_norm[1] = a.G_norm;
    G_norm[2] = -b.G_norm;
    G_norm[1] - G_norm[2] = a.dG;
    mdot[1] = a.mdot;
    mdot[2] = -b.mdot;
    edot[1] = a.q_conv;
    edot[n + 1] = -b.q_conv;
    for i in 2:n loop
      edot[i] = if mdot[1]-dM[1]*(i-1) > 0
        then (mdot[1]-dM[1]*(i-1))*h[i-1]
        else (mdot[1]-dM[1]*(i-1))*h[i];
    end for;
    edot[n+1] = if mdot[2]>0 then mdot[2]*h[n] else mdot[2]*b.h;
    dM = (mdot[1] - mdot[2])*ones(n)/n;
    for i in 1:n loop
      dU[i] = edot[i] - edot[i + 1] - p[1]*der(V[i]) + Q_s[i];
    end for;
  end BalanceTwoPortSingleSpecial;

  partial model ThermalModelSpecial
    replaceable model Medium = StateVariables_p1hn ;
    extends Medium;
  protected
    Real km[n],km_mean;
    Real kp[3];
    Real kh[3, n];
  equation
    for i in 1:n loop
      km[i] = V[i]*(pro[i].ddph*d[i] + pro[i].ddhp);
      kh[1, i] = 1 - h[i]*pro[i].ddph;
      kh[2, i] = pro[i].ddph;
      kh[3, i] = pro[i].ddph*p[1] - d[i];
    // state equations
    km[i]*der(h[i]) =
      kh[1, i]*dM[i] + kh[2, i]*dU[i] + kh[3, i]*der(V[i]);
  end for;
  km_mean = sum(V)*(pro[n + 1].ddph*d_mean + pro[n + 1].ddhp);
  kp[1] = d_mean + h_mean*pro[n + 1].ddhp;
  kp[2] = -pro[n + 1].ddhp;
  kp[3] = -d_mean*d_mean - pro[n + 1].ddhp*p[1];
  km_mean*der(p[1]) =
    kp[1]*sum(dM) + kp[2]*sum(dU) + kp[3]*sum(der(V));
  M[i] = d[i]*V[i];
  U[i] = pro[i].u*M[i];
  // Mean value of enthalpies, gives mean thermal state,
  //   use sum(d) for total mass to avoid algebr. loop
  h_mean = h[1:n]*d[1:n]/sum(d[1:n]);
end ThermalModelSpecial;

model FlowModelTwoPortSingleSpecialDyn
  "Lumped FM for use with distributed thermal model"
  extends CommonRec.ConnectingVariablesSingleDynamic;
  extends CommonRec.BaseGeometryVars(alpha=0);
  extends BalanceSetSingleSpecial;
  replaceable model PLoss = CommonRec.PressureLossDistributed;
  extends PLoss;
equation
  // This equation is general:
  dz = L/n;
  G_norm[2] = if mdot[2] > 0
    then mdot[2]*mdot[2]/d_mean/A else -mdot[2]*mdot[2]/ddown/A;
  dG = G_norm[1]-G_norm[2] + dGdown;
  // This is the momentum balance equation
  L*der(mdot[2]) = dG + (p[1] - pdown)*A - sum(Ploss)/n*L*Dhyd*Pi;
end FlowModelTwoPortSingleSpecialDyn;

model WaterSteamSpecial
  extends StateVariables_p1hn;
  parameter Integer mode=0;
  Integer phase[n + 1];
equation
  // Region check with events, only covers regions 1, 2 and 4
  for i in 1:n loop
    phase[i] = if (mode == 4) then 2 else (if (mode>0) then 1 else
      (if ((h[i] < SteamIF97.hlofp(p[1]))
        or (h[i] > SteamIF97.hvofp(p[1]))) then 1 else 2));
    pro[i] = Water.water_ph(p[1],h[i],phase[i],mode);
  end for;
  phase[n+1] = if (mode == 4) then 2 else (if (mode>0) then 1 else
    (if ((h_mean < SteamIF97.hlofp(p[1]))
      or (h_mean > SteamIF97.hvofp(p[1]))) then 1 else 2));
  pro[n+1] = Water.water_ph(p[1],h_mean,phase[n+1],mode);
end WaterSteamSpecial;
```

```
  model ControlVolumeTwoPortSingleSpecial "Two port volume"
    extends BalanceTwoPortSingleSpecial;
    extends ThermalModelSpecial;
    parameter SIunits.Volume V0(start=1) "Constant volume";
    parameter SIunits.Length L0(start=1.0) "Tube length";
    parameter SIunits.Length Dhyd0(start=1.0) "Hydraulic diameter";
    parameter SIunits.Area A0(start=1.0) "Flow area";
    extends FlowModelTwoPortSingleSpecialDyn
      (L=L0, Dhyd=Dhyd0, A=A0);
  equation
    V = V0*ones(n)/n;
  end ControlVolumeTwoPortSingleSpecial;

  model PipeFrictionSpecial
    extends CommonRec.PressureLossDistributed;
    parameter Real k(start=0.5) "Friction coefficient";
    SIunits.MassFlowRate mdot[2];
    SIunits.Density d[n];
    SIunits.Area A;
  equation
    for i in 1:n loop
      Ploss[i] = k*abs(mdot[2])*mdot[2]/(2*A*A*d[i]);
    end for;
  end PipeFrictionSpecial;

  model PipeSpecial "Special pipe model w 1 pressure and n temps."
    parameter SIunits.Pressure p0=1.1e5;
    parameter SIunits.SpecificEnthalpy h0=3.0e5;
    parameter SIunits.MassFlowRate mdot0=0.1;
    extends ControlVolumeTwoPortSingleSpecial(
      redeclare model PLoss=PipeFrictionSpecial,
      redeclare model Medium=WaterSteamSpecial,
      V0=1.0, L0=1.0);
  end PipeSpecial;
end SpecialPipe;

model SurgeTank
  extends Balances.TwoPortLumpedAdiabatic;
  extends Water.WaterSteamMedium_ph;
  parameter SIunits.Pressure p0=1e5;
  parameter SIunits.Volume V0=0.01;
equation
  der(V[1]) = dM[1]/d[1];
  // Approximately der(d)=0
  p[1]*(2*V0 - V[1]) = p0*V0;
  der(h[1]) = dU[1]/M[1];
  W_t[1] = 0;
end SurgeTank;

model ConstantFlow
```

```
  extends
    Modelica.ThermoFlow.PartialComponents.Valves.FlowModelBaseSingle(
      redeclare Interfaces.SingleDynamic.FlowB a,
      redeclare Interfaces.SingleDynamic.FlowB b);
  parameter SIunits.MassFlowRate mdot0=0.15;
  parameter SIunits.Area A=0.1;
equation
  a_upstream = true;
  mdot = mdot0;
  a.G_norm = if a_upstream then a.mdot*a.mdot/a.d/A
                           else -a.mdot*a.mdot/b.d/A;
  b.G_norm = -a.G_norm;
end ConstantFlow;

model BoilerPipe
  Modelica.ThermoFlow.Components.Water.PipesAndVolumes.PipeGeometry geo;
  SpecialPipe.PipeSpecial HeatPipe(n=10, h0=geo.h0, mdot0=geo.mdot0,
    k=geo.k, V0=geo.V, L0=geo.L, Dhyd0=geo.D, A0=geo.A);
  Modelica.ThermoFlow.Components.Water.Reservoirs.WaterResD_ph Sink;
  Modelica.ThermoFlow.Components.Water.Reservoirs.WaterResD_ph ResIn;
  Modelica.ThermoFlow.Components.HeatFlow.Sources.HeatD HeatD1(n=10);
  ConstantFlow Flow1(A=geo.A);
  SurgeTank Source;
  Modelica.ThermoFlow.Components.SingleDynamic.Valves.LinearValve
    Valve1(A=geo.A);
equation
  connect(HeatD1.qa, HeatPipe.q);
  connect(HeatPipe.b, Sink.a);
  connect(Valve1.b, HeatPipe.a);
  connect(Source.b, Valve1.a);
  connect(Flow1.b, Source.a);
  connect(ResIn.a, Flow1.a);
end BoilerPipe;
```

Paper IV

Parameter Optimization of a Non-linear Boiler Model

Jonas Eborn and James Sørlie

Abstract

The object of this study is a steam generation process model developed by Åström and Bell. The paper reports improvements obtained by tuning uncertain physical parameters as well as a verification of the model structure. Employing measurement data and methods from system identification, the paper demonstrates the complementary nature of first-principle modeling and identification. Results show that statistical methods, combined with a systematic search strategy, allow improvement of a larger number of parameters than is possible through manual tuning.

Keywords: control systems, nonlinear modeling, optimization, parameter estimation, system identification.

Reproduced from

Eborn, J. and J. Sørlie (1997): “Parameter optimization of a non-linear boiler model.” In Sydow, Ed., *15th IMACS World Congress*, vol. 5, pp. 725–730. W&T Verlag, Berlin, Germany.

1. Introduction

Models built upon first principles are useful for both analysis and design. The physical relevance and insight they provide often reveal structural properties of the actual process, something which black-box models cannot do. With first-principle models there are frequently parameters which are uncertain or even impossible to determine without detailed measurements. Often, to achieve a good visual fit to measured data, trial-and-error techniques are used to adjust parameters manually. In this paper, we study the use of system identification applied to first-principle models; cf. Bohlin (1991). Our aim is to demonstrate a systematic approach to the task of parameter tuning.

As a case study, we use the third and fourth-order non-linear implicit differential equation models developed by Åström and Bell (1988; 1993; 1996). The mathematical model has a relatively small number of physical parameters and input conversion factors. Most of the physical parameters are well determined from construction data. However, some parameters are obtained by rough estimates, like metal masses and a friction factor in the flow through the down-comers/risers circuit. The friction factor has been manually adjusted to achieve good simulation results. It is of interest to see what improvements can be obtained thru optimization of these uncertain parameters.

Software tools that have been used for this study are the modeling environment OmSim, see Andersson (1994), and the grey-box identification tool-kit IdKit, see Graebe (1990); Sørlie (1996). Motivating this paper is our desire to demonstrate the complementary nature of modeling and identification, as well as the need for software tools integrating modeling and simulation with parameter optimization of general non-linear first-principle models.

2. Model Definition

The details of the model's derivation are given in Åström and Bell(1988; 1993; 1996); here, we briefly survey their results. Printing provisions prohibit including all the modeling equations. They have been programmed in Omola, see Andersson (1994); Sørlie and Eborn (1997), and are available upon request from the authors.

An idealized physical model for the system is shown in Figure 1. Steam vapor is vented from the drum with flow-rate q_s. Feed-water enters the drum in a sub-cooled liquid state with flow-rate q_{fw} and temperature T_{fw}. Steam vapor is generated by channeling the liquid phase from the drum through a down-comers/risers circuit. The heat flow-rate Q into the

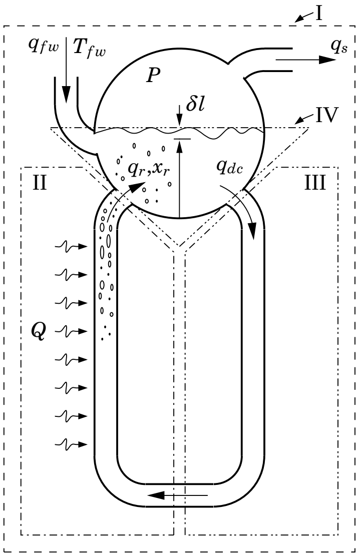


Figure 1 Ideal physical model of a steam generation process.

risers comes from the combustion of fuel. The flow-rate into the circuit q_{dc} is driven by the density gradient caused by the phase change in the risers. At the risers outlet, the two-phase mixture is characterized by the mass flow-rate q_r and vapor mass-fraction x_r.

The fundamental modeling simplification is that the two phases of water inside the system are everywhere in a saturated thermodynamic state. With this assumption, all thermodynamic properties can be characterized by one independent variable. The drum pressure P is chosen to be this key state variable since it is the most globally uniform variable in the system. Another key assumption is an instantaneous and uniform thermal equilibrium between water and metal everywhere. This simplifies including thermal capacitance effects.

Indicated in Figure 1 are the boundaries of four thermodynamic control volumes. Mass and energy balances for the global control volume (c.v. I) yield two state equations. The state variables are pressure P and the total volume of liquid water in the system V_{wt}. By combining the mass and energy balances for c.v. II to eliminate the flow-rate q_r, a third state equation is derived with the vapor mass-fraction x_r as state variable. By considering fluid friction in c.v. III, a fluid momentum balance establishes the flow-rate q_{dc}. A combination of the mass and energy balances for c.v. IV yields a fourth state equation with state variable V_{sd}, the volume of steam vapor below the liquid surface. Assembled in matrix notation, the fourth-order model structure (i. e., a set of implicit differential state equations) is:

$$\mathcal{M}_4: \quad \begin{bmatrix} e_{11} & e_{12} & 0 & 0 \\ e_{21} & e_{22} & 0 & 0 \\ 0 & e_{32} & e_{33} & 0 \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial t} V_{wt} \\ \frac{\partial}{\partial t} P \\ \frac{\partial}{\partial t} x_r \\ \frac{\partial}{\partial t} V_{sd} \end{bmatrix} = \begin{bmatrix} q_{fw} - q_s \\ Q + h_{fw} q_{fw} - h_s q_s + \Delta_{\mathrm{I}} \\ Q - h_c x_r q_{dc} + \Delta_{\mathrm{II}} \\ \frac{V_{sd} - V_{sd}^0}{\tau_{sd}} + \frac{(h_{fw} - h_w)\, q_{fw} + \Delta_{\mathrm{IV}}}{\rho_s h_c} \end{bmatrix} \qquad (1)$$

The elements of the coefficient matrix e_{11}, e_{12}, e_{21}, etc., are state dependent. The complexity of these expressions prohibits including them here; see Sørlie and Eborn (1997). On the right, Δ_{I}, Δ_{II} and Δ_{IV} represent under-modeling, i. e., unmodeled energy interactions (nominally taken to be zero). The initial state conditions are parameterized $[V_{wt}^0, P^0, x_r^0, V_{sd}^0]^T$. In addition to these, the model involves seven physical parameters: metal masses m_d, m_r, m_{dc}, volumes V_d, V_r, V_{dc}, and a fluid friction coefficient in the down-comers k. Known constants are the specific heats C_{fw} and C_p for the feed-water and metal respectively.

For the purpose of level control, Bell and Åström (1996) proposed a variational measurement model for the liquid level in the drum:

$$\delta l = \left(\left(V_{wd} - V_{wd}^0 \right) + \left(V_{sd} - V_{sd}^0 \right) \right) / A_d. \qquad (2)$$

The level variation δl is caused by variations in the volumes of liquid in the drum V_{wd}[1] and the steam below the surface V_{sd}. This model introduces one additional physical parameter: A_d, the drum's cross-sectional area at the nominal level. The aim of including variation in V_{sd} is to capture the level dynamics known as the "shrink-and-swell" effect; cf. Bell and Åström (1996).

To assess the necessity of including the fourth state equation in $\mathcal{M}_4$, we shall investigate parameter optimization of both third and fourth-order model structures. In the third-order structure $\mathcal{M}_3$, the state variable V_{sd} in (2) is replaced with an instantaneous value. Engineering judgment suggests several approximations for its value. The fourth-order structure

[1] $V_{wd}(t) = V_{wt}(t) - V_{dc} - (1 - \alpha_r(t))\, V_r$ where α_r is the total volume fraction of steam in the risers, i. e., $\alpha_r = V_{sr}/V_r$; an approximation with form $\alpha_r \approx \mathrm{fcn}(P, x_r)$ is given in Åström and Bell (1988); Åström and Bell (1993).

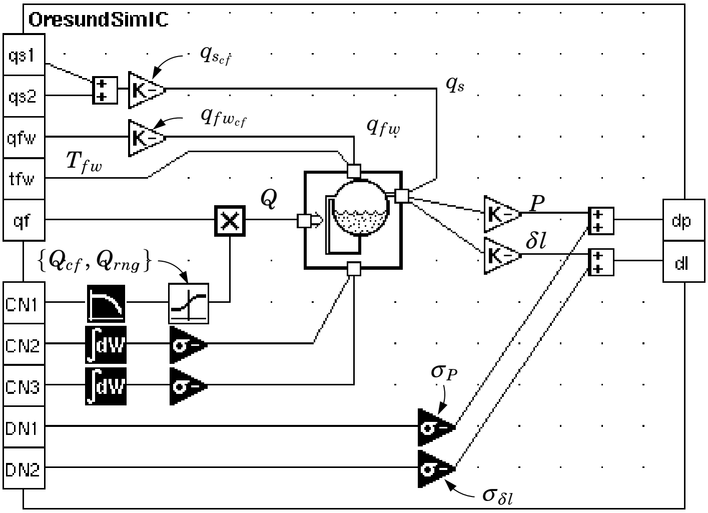


Figure 2 Omola definition of the simulation interface to signals of the experimental data, and stochastic input and output-error models.

$\mathcal{M}_4$ involves a similar set of hypotheses for the bubble-residence time-constant τ_{sd}. The heuristics for the values which have been tested are:

$$V_{sd} = \begin{cases} b_1 & \text{hyp. 0,} \\ b_1 \alpha_r V_r & \text{hyp. 1,} \\ b_1 x_r q_r & \text{hyp. 2,} \end{cases} \quad \tau_{sd} = \begin{cases} \dfrac{b_1 \rho_s}{q_s} \left(V_d - V_{wd} \right) & \text{hyp. 0,} \\ \dfrac{b_1 \rho_s}{x_r q_r} \left(2V_{sd}^0 - V_{sd} \right) & \text{hyp. 1.} \end{cases}$$

In each model structure, b_1 is a “grey-box” parameter to be optimized.

For parameter optimization, we use three datasets obtained from experiments reported in Åström and Eklund (1972). Figure 2 shows the simulation interface to the five measured inputs: two steam flows, feed-water flow, feed-water temperature and fuel flow. An interface with real data necessitates conversion factors; the simulation schematic shows several. Most uncertain is the calibration of the heat input Q. Because the chemical energy content of the fuel is known to vary, this gain has been probabilistically modeled with a nominal value Q_{cf} and a known, bounded range Q_{rng}. More certain are the calibrations of the steam mass flow-rates $q_{s_{cf}}$ and feed-water mass flow-rate $q_{fw_{cf}}$. Assuming liquid flow measurements are more precise than vapor flow measurements, we shall consider the later[2] a known constant. This leaves as additional parameters for

[2] *Correction:* The word “later” here refers to $q_{fw_{cf}}$ and not vapor flow measurements.

optimization Q_{cf} and $q_{s_{cf}}$.

In addition to the stochastic modeling of the gain Q_{cf}, the simulation interface includes simple stochastic input and output-error models. The focus of this paper is parameter optimization in a deterministic setting; accordingly, on the instantaneous output-error models will be investigated here. In Sørlie and Eborn (1997), the input-error models are used to investigate the effects of under-modeling, i. e., Δ_{I} etc. Summarizing the model definition, we have third and fourth-order model structures $\mathcal{M}_3$ and $\mathcal{M}_4$ with parameterization $[m_d,\ m_r,\ m_{dc},\ V_d,\ V_r,\ V_{dc},\ k,\ A_d,\ b_1,\ Q_{cf},\ q_{s_{cf}}]$.

3. Parameter Optimization

Parameter estimation has been investigated with the third and fourth-order model structures $\mathcal{M}_3$ and $\mathcal{M}_4$. The three datasets involve perturbation on different inputs; steam flow, fuel flow and feed-water flow. The IDKIT software, see Graebe (1990), utilizes a gradient search method to minimize the likelihood function for the given observations. Along with the parameter estimates, the software also calculates values of the Akaike Information Criterion (AIC) and the loss function; these values are useful for comparisons and hypothesis testing. The search method requires good initial guesses for the parameters. Reasonable values were obtained from Åström and Bell (1993; 1996). Notationally, we denote a nominally parameterized model $M_3(\Theta_0)$. Estimated models are named in a similar fashion; e. g.,, $M_3(\hat{\Theta}_1)$ is the model obtained from dataset 1 with the third-order model structure.

Choosing free parameters: Augmentation and over-parameterization

The modeling goal of this study was to obtain a good deterministic model. Nevertheless, an error description is very important for the optimization. Purely deterministic models can not explain everything in the data. In optimization, this leads to convergence problems. By first estimating measurement error variances with nominal physical parameters and subsequently augmenting the free parameter space, a good deterministic model is obtained. The estimated error variances give a measure of the model uncertainty. The principle used when choosing free parameters (i. e., free for search) is to start with the parameters that are least known or have a large impact on outputs. In our case this means error variances, input conversion factors and some initial values. Then, augmentation to include hypothesis testing and optimization of physical parameters can be done.

When choosing what parameters to optimize, over-parameterization is an important issue. This is very common in physical models based on first

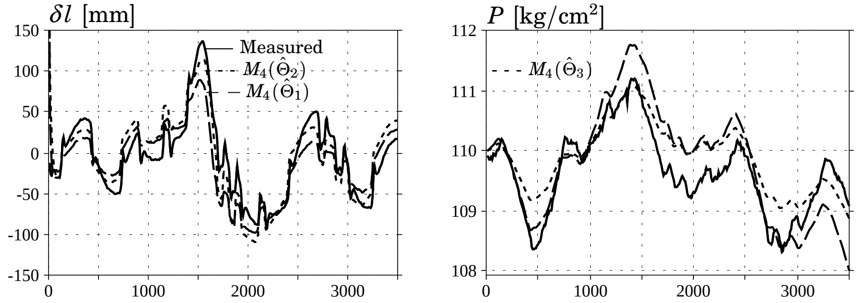


Figure 3 Model obtained from dataset 1 validated against drum-level from dataset 2 and pressure from dataset 3. Compared also with the best model for each dataset.

principles. Energy storage in metal e. g., depends only on the product mC_p; thus these parameters can not be estimated independently. More subtle over-parameterizations may be overlooked. In this study we set out to estimate the friction factor, k. It was deemed impossible since k mainly affects offset in the drum level, which makes it coupled to the initial value x_r^0.[3] The coupling can be seen by examining the parameter sensitivities, i. e., the Hessian of the likelihood function. This reveals cross-couplings between such parameters. Evaluation of estimated models with different, fixed values for k show that the model behavior is very insensitive to changes in k. In fact, the AIC value is least for the nominal value k=0.005.

Estimates for third-order model structure

The main result from running optimization on the third-order model is that the total mass, $m = m_r + m_d + m_{dc}$, is estimated to be 500 tons, considerably more than the nominal value 300 tons. The difference is not surprising though since the nominal value was obtained by rather rough calculations. The estimate is consistent with the first two datasets, while the third dataset gives estimates considerably higher, $m > 1000$ tons. This result is less reliable though, since the excitation in this dataset is very small and time variations in Q_{cf} has a considerable impact on the pressure, which is the output affected by m; see Figure 3.

Testing the different hypotheses concerning V_{sd} reveals that hypothesis 2 is considerably better in conjunction with the first dataset, yielding AIC=2920 compared to 2980 and 3080 for hypotheses 1 and 0 respectively. This is not the case when applied to the other datasets. Optimization suppresses the influence of V_{sd} by reducing the factor b_1; values of the Akaike

[3] Assuming the system was in near equilibrium during the experiments, the equilibrium solution for the third state equation can be used to parameterize x_r^0.

Dataset:	1. Steam flow				2. Fuel flow				3. Feed-water flow			
	m	σ_P	$\sigma_{\delta l}$	AIC	m	σ_P	$\sigma_{\delta l}$	AIC	m	σ_P	$\sigma_{\delta l}$	AIC
Model	[ton]	[bar]	[mm]		[ton]	[bar]	[mm]		[ton]	[bar]	[mm]	
$M_3(\Theta_0)$	300	0.9	50	3494	300	1.3	43	3634	300	0.63	74	3484
$M_3(\hat{\Theta})$	542	0.8	27	2920	501	1.2	30	3304	1080	0.3	20	2046
$M_4(\hat{\Theta})$	460	0.8	10	2217	392	1.2	20	3030	1054	0.3	10	1622
Cross-validation of $M_4(\hat{\Theta}_1)$					460	1.3	23	3174	460	0.5	10	1885

Table 1 Estimation results for different model structures and datasets.

criterion are almost exactly the same with different hypotheses. This suggests that the simple hypotheses are insufficient and there are additional dynamics concerning V_{sd}. These are introduced as a state in the fourth-order model.

Estimates for fourth-order model structure

Results from the fourth-order model are consistent with the previous results in that the estimates of the total mass give similar results and the additional dynamics give an appreciable contribution; drum level error and AIC values decrease according to Table 1.

With the additional dynamics in the fourth-order structure there are also possibilities to optimize other parameters. Besides total mass, also drum and riser mass influence the behavior. These have been estimated on the first dataset and were found to be: m_d=61 tons, m_r=272 tons. These figures are reasonable relative to each other and the total mass. Also the grey-box factor scaling the bubble time constant τ_{sd} was estimated to be b_1=1.9. To check the validity of the results this model was used on the other datasets after estimation of only conversion factors and x_r^0. It was also compared to the best possible model for those datasets. Comparisons can be seen in Table 1 and in Figure 3.

The validated model performs almost as good as the best ones for each dataset. In the fuel-flow data there seems to be an overshoot phenomenon in drum-level not caught by the model, this explains why the model optimized on the first dataset gives a higher AIC value. In the third dataset there is the problem with low excitation and time-varying Q_{cf} mentioned earlier.

4. Structure determination

The task of structure determination is supported by having modeling and simulation tools integrated with optimization/identification tools. In this study we have used OmSim to create different model structures and hypotheses and test them in simulation to see their qualitative behavior. The model equations are then exported to IdKit for parameter optimization and hypothesis testing. Statistical measures like the AIC give an objective evaluation of the model structures. This together with the subjective measure obtained in simulation provide the necessary information whether to accept or reject a model structure.

In this case study two different model structures were compared; the third and fourth-order models described in previous sections. The statistical measures given in Table 1 all show that the fourth-order model describes drum-level much better and this is confirmed in the simulations shown in Figure 4, see the differences in drum-level behavior at t =1100 and 1700 seconds. The attempt at cross-validation of the fourth-order model could be confusing just looking at the AIC values since in dataset 3 the AIC=1622 for the best model is much lower than 1885. But in simulations it can be seen that this mainly depends on deficiencies in the experiment, e. g., time-varying conversion factors. Qualitatively, the model from the first dataset $M_4(\hat{\Theta}_1)$ behaves better than the statistically 'best' model; in Figure 3, the increased mass estimates in $M_4(\hat{\Theta}_3)$ effectively flatten the pressure variations and inadvertently suppress dynamics present in $M_4(\hat{\Theta}_1)$.

A close integration of modeling and identification tools also makes it easier to test different model hypotheses. For the third-order model this was reported in the previous section. In the fourth-order model there

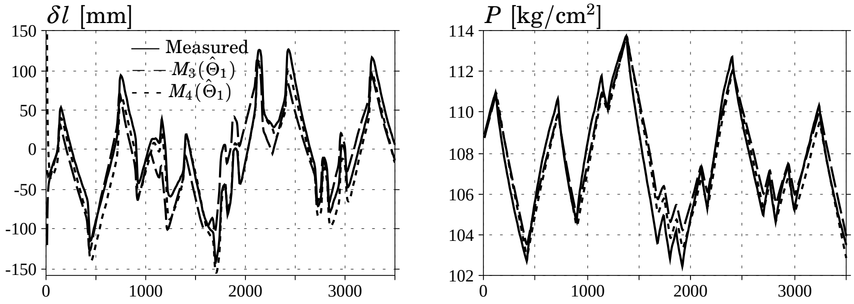


Figure 4 Simulations of third and fourth-order models compared to data from dataset 1.

Dataset:	1. Steam flow				2. Fuel flow				3. Feed-water flow			
Hypoth.	b_1	σ_P	$\sigma_{\delta l}$	AIC	b_1	σ_P	$\sigma_{\delta l}$	AIC	b_1	σ_P	$\sigma_{\delta l}$	AIC
0	0.5	0.8	14	2476	0.22	1.2	27	3232	0.43	0.34	14	1859
1	1.9	0.8	10	2220	2.4	1.2	20	3030	1.9	0.34	10	1622

Table 2 Evaluation of hypotheses of the fourth-order model structure.

are two hypotheses concerning bubble residence time, τ_{sd}. These have been tested in favor of hypothesis 1; see Table 2. AIC values for all three datasets are lower and the scaling factors b_1 are all close to the same value, which favors hypothesis 1.

5. Conclusions

This paper reports on parameter estimation on two different non-linear model structures for a drum-boiler process. The results verify that the fourth-order structure better describes the complicated drum-level dynamics. A large number of uncertain physical parameters have been estimated, a task which would have been impossible by commonly used trial-and-error testing. This is especially true since uncertain input conversion factors introduce drift in simulations. Using a search strategy to first estimate conversion factors and error variances, and then iteratively augment free parameter space, up to 10 parameters can be simultaneously optimized without difficulty. Statistical measures of the model fidelity, together with qualitative information obtained in simulation, give the information needed to assess different model hypotheses.

We believe that the dual nature of modeling and identification demonstrated here is very powerful and should be supported by well integrated software tools. For linear model structures such tools exist. In the nonlinear case much remains to be done.

Acknowledgments

This work was supported by a research grant from Sydkraft AB.

References

Åström, K. J. and R. D. Bell (1988): “Simple drum-boiler models.” In *IFAC Int.Symposium on Power Systems, Modelling and Control Applications*. Brussels, Belgium.

Åström, K. J. and R. D. Bell (1993): “A nonlinear model for steam generation processes.” In *Preprints IFAC 12th World Congress*. Sydney, Australia.

Andersson, M. (1994): *Object-Oriented Modeling and Simulation of Hybrid Systems*. Ph.D. thesis TFRT–1043–SE, Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden.

Åström, K. and K. Eklund (1972): “A simplified non-linear model of a drum boiler-turbine unit.” *Int. J. Control*, **16:1**, pp. 145–169.

Bell, R. D. and K. J. Åström (1996): “A fourth order non-linear model for drum-boiler dynamics.” In *IFAC’96, Preprints 13th World Congress*, vol. O, pp. 31–36. San Francisco, CA.

Bohlin, T. (1991): *Interactive System Identification: Prospects and Pitfalls*. Springer-Verlag, Berlin, Germany.

Graebe, S. F. (1990): *Theory and Implementation of Gray Box Identification*. Ph.D. thesis TRITA–REG–9006, Royal Institute of Technology, Dept. of Automatic Control, Stockholm, Sweden.

Sørlie, J. and J. Eborn (1997): “A grey-box identification case study: The Åström–Bell drum-boiler model.” Technical Report ISRN LUTFD2/TFRT--7563--SE. Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden.

Sørlie, J. A. (1996): *On Grey-Box Model Definition and Symbolic Derivation of Extended Kalman Filters*. Ph.D. thesis TRITA–REG–9601, S3–Automatic Control, Royal Institute of Technology, Stockholm, Sweden.

LUND INSTITUTE OF TECHNOLOGY
Lund University

ISSN 0280-5316
ISRN LUTFD2/TFRT--1061--SE