Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems

Von der Fakultät für Maschinenbau
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von: Christoph C. Richter
aus: Braunschweig

eingereicht am: 11. Januar 2008
mündliche Prüfung am: 3. April 2008

Referenten: Prof. Dr.-Ing. Jürgen Köhler
Prof. Dr.-Ing. Gerhard Schmitz
Vorsitzender: Prof. Dr.-Ing. Martin Mönnigmann

2008

Preface

This thesis can be regarded as a follow-up project to the PhD thesis of Tegethoff (1999) moving from his C++ simulation platform to a component model library written in Modelica complemented by a fluid property library developed in C/C++ with interfaces to various software tools and programming languages including Modelica. The thesis belongs somewhere in the intersection of Computer Science, Systems Design Engineering, and Thermodynamics. It examines the advantages and possible pitfalls of object-oriented model library design using the free object-oriented modeling language Modelica. During the time of the library development, I was involved in the design of parts of the Modelica Standard Library and got very good insight into the language design itself by being a member of the Modelica Association. The enthusiasm of the members of the Modelica Association and the Modelica Fluid Group was a continuous source of motivation and a great help with a lot of problems that emerged during the library design process.

Acknowledgments

Writing a thesis is an iterative process that requires continuous input from other people especially when developing simulation libraries that are meant to be used by others than the author himself. I would not have been able to finish this project without the strong support of a number of people to whom the following paragraphs are dedicated. The order is random and not meant to be a ranking of any sort.

First I wish to thank all postgraduate students at the Institut für Thermodynamik for their support and help in many situations. Special thanks to Roland who was never bored with my endless questions regarding C++ and Qt and with whom I was able to develop a program as unique as the StateViewer. I would also like to thank Christine, Julia, Christian S., Roland, and Niko for many nice evenings including sushi, wine, and memorable Cranium sessions. Special thanks to Christine for cheering me up in the mornings by having a little chat before everyone else showed up. Also writing chapter 6 would not have been possible without her strong support that I really appreciated. Thanks to Kai and Marcos who had to test and extend my libraries for not getting sick of the many updates that they had to adopt their models to. Special thanks also to Christian T. for his strong support with the ejector test stand and for his never-ending effort to keep up with me when going for a run after a long day of work.

I also want to thank Willi who was always keen enough to discuss new ideas and who did not stop me when starting to redesign `TILFluids` from scratch in late spring 2006. I am also very grateful that Willi taught me how to use Modelica in an in-house course knowing the language just a little better than the participants of the course back then. This thesis would not have been possible without his strong support especially during the last weeks and days before finishing it.

Furthermore, I want to thank the staff at TLK for utilizing some of the software I developed despite all the bugs. Many thanks also to the master students at the Institut für

Thermodynamik that helped me with some aspects of my work, especially to Martin for his tool ModelicaCDV (which I misspelled many times as ModelicaCVD) and Nils for his implementation of heat exchangers in Modelica.

I am grateful to the members of the Modelica Association for answering the really tricky Modelica questions. I especially want to thank Katrin Prölß, Hubertus Tummescheit, Francesco Casella, and the entire Modelica Fluid Group for many fruitful discussions and nice evenings at the Modelica Design Meetings.

I am glad to express my sincere gratitude to Prof. Köhler for supervising this thesis and to Prof. Schmitz for his valuable inputs. I also want to thank the technical staff at the Institut für Thermodynamik for their support during the three years I spent there.

Special thanks to the Canada gang for the yearly meetings that always were a source of motivation to me. Thanks Debbie, Jenny, Sven, and Matthias for the many outdoor events and cooking sessions. Also thanks for changing me from a couch potato into a marathon finisher.

Special thanks go to my girlfriend Frederike for always being supportive even during the hard times of self-doubts. The two months in New Zealand are unforgettable and belong to my most precious memories. Thanks also to Frederike, Mandy, and Christian T. for the nice evenings taking dancing lessons. Also many thanks to my parents for their loving support and confidence in me and to my brother Marius for proofreading this thesis and for many motivating phone calls.

For the financial support, I want to thank the Studienstiftung des deutschen Volkes that was brave enough to support me since my second semester as an undergraduate student all the way until finishing my doctoral thesis.

Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems

Abstract

This thesis proposes two new model libraries for fluid properties and for components that can be used for the simulation of thermodynamic systems such as refrigeration, air-conditioning, and heat-pump systems. The new fluid property library is written in C/C++ and can be interfaced from various software tools and programming languages. The new component model library is written in the object-oriented equation-based modeling language Modelica. Furthermore, tools for the automated generation of class diagrams and the visualization of the solution process as well as the numerical results in relevant diagrams such as pressure-enthalpy diagrams are presented.

Both new libraries are based on a thorough object-oriented analysis. Graphical representations for all object-oriented features of Modelica based on the elements defined in the Unified Modeling Language are introduced. A set of general design rules for the development of object-oriented component model libraries is formulated to ensure that the resulting library can be used by the entire spectrum of possible users from experienced developers to design engineers.

The new object-based fluid property library is based on a generalized approach to include external fluid property computation codes in Modelica. It is simple to extend to additional external fluid property computation codes. It allows for a numerically efficient handling of fluid properties in Modelica and in a number of software tools. The numerical efficiency of the presented approach matches the numerical efficiency of available fluid property computation codes formulated in Modelica.

The new model library for components and systems was developed based on the newly introduced design rules. It features a structure that is simple to understand and flexible to allow for extensions. All balance equations are formulated in an easy and comprehensible way in base components. The new component model library contains models with different levels of detail to allow for a problem-dependent model selection.

Two applications are presented to demonstrate the capabilities of the two new libraries. A prototype Peltier heat exchanger heating one water stream while cooling another one is analyzed during a transient reversion of the applied voltage. The second example analyzes a prototype ejector refrigeration system partly developed within the scope of this thesis. It is shown that an ejector improves the COP of a CO_2 refrigeration system. The two selected examples demonstrate the extendibility and multidisciplinarity of the new libraries.

Both libraries are already used and extended by a team of developers at the Institut für Thermodynamik and the TLK-Thermo GmbH.

Vorschlag für neue objektorientierte gleichungsbasierte Modellbibliotheken für thermodynamische Systeme

Kurzfassung

Diese Arbeit beschreibt zwei neue Modellbibliotheken mit Stoffdaten- und Komponentenmodellen zur Simulation thermodynamischer Systeme wie zum Beispiel Kälteanlagen, Klimaanlagen und Wärmepumpen. Die neue Stoffdatenbibliothek ist basierend auf C/C++ entwickelt worden und verfügt über Schnittstellen für unterschiedliche Anwendungsprogramme und Programmiersprachen. Die neue Komponentenbibliothek ist in der objektorientierten gleichungsbasierten Modellierungssprache Modelica geschrieben. Es werden außerdem Werkzeuge vorgestellt, die die automatische Generierung von Klassenstrukturdiagrammen und die Visualisierung des Lösungsprozesses sowie der numerischen Ergebnisse in thermodynamischen Diagrammen wie zum Beispiel dem p,h-Diagramm ermöglichen.

Beide neuen Bibliotheken basieren auf einer gründlichen objektorientierten Analyse. Um diese einheitlich zu ermöglichen, werden grafische Darstellungen für alle objektorientierten Merkmale von Modelica basierend auf der UML eingeführt. Es werden des Weiteren allgemeingültige Richtlinien für die Erstellung von objektorientierten Modellbibliotheken entwickelt, die sicherstellen, dass die entwickelte Bibliothek das gesamte Spektrum möglicher Benutzer vom Code-Entwickler bis hin zum Anwender unterstützt.

Die neue objektbasierte Stoffdatenbibliothek basiert auf einem generalisierten Ansatz zur Einbindung externer Stoffdatenbibliotheken in Modelica. Der beschriebene Ansatz lässt sich einfach auf weitere externe Bibliotheken zur Stoffdatenberechnung erweitern. Er erlaubt die numerisch effektive Behandlung der eingebundenen Bibliotheken in Modelica sowie in Anwendungsprogrammen. Der generalisierte Ansatz ist vorhandenen Modelica-internen Stoffdatenbibliotheken in numerischer Hinsicht ebenbürtig.

Die neue Modellbibliothek für Komponenten und Systeme verfügt über eine Struktur, die einfach verständlich und flexibel erweiterbar ist. Alle Erhaltungsgleichungen sind in einfach verständlicher Form in Basiskomponenten formuliert. Die neue Komponentenmodellbibliothek enthält Modelle unterschiedlicher Modellierungstiefe, um eine problemabhängige Auswahl von Modellen zu erlauben.

Zwei Anwendungen für die beiden neuen Modellbibliotheken werden präsentiert. Die erste Anwendung ist die transiente Simulation einer Peltier-Wasser-Wasser-Wärmeübertragers während einer Spannungsumkehrung. Das zweite Beispiel ist ein Ejektor-Kältekreislauf, der teilweise im Rahmen dieser Arbeit entwickelt wurde. Es wird gezeigt, dass der Einsatz eines Ejektors den COP eines CO_2-Kältekreislaufes steigert. Beide Anwedungsbeispiele demonstrieren die Erweiterbarkeit und die Multidisziplinarität der neuen Bibliotheken.

Beide neuen Modellbibliotheken werden von einem Team von Anwendern und Entwicklern am Institut für Thermodynamik und bei der TLK-Thermo GmbH genutzt und erweitert.

Contents

Chapter 1

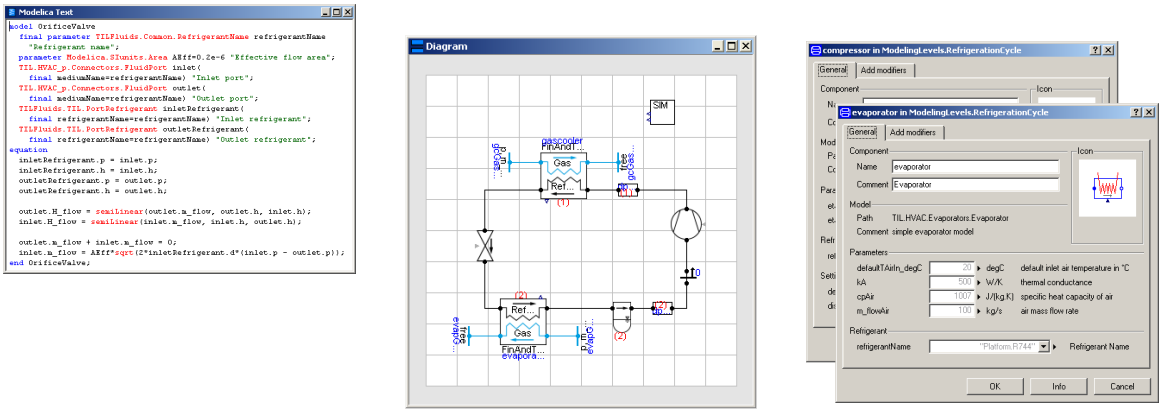
Introduction

Computer simulations are one of the foundations of modern engineering. A lot of engineering knowhow is coded into system models that allow for a deeper understanding and reliable predictions of system behavior. This chapter motivates the development of a new object-oriented Modelica library to simulate and analyze thermodynamic systems and provides background information on the development of modern modeling languages especially of Modelica.

1.1 Motivation

Over the past decades, computer simulations have become a central foundation of modern engineering. A large number of simulation tools has been developed to solve a wide range of engineering problems. This thesis focuses on the analysis of general strategies for the development of object-oriented component model libraries and proposes suitable basic structures. It presents a new model library to compute fluid properties as well as a new component model library written in the free modeling language Modelica. The component models are either zero-dimensional, i.e., do not take into account spatial distributions, or one-dimensional, i.e., take into account spacial distributions in one spatial direction. The idea for the new object-oriented component model library emerged from the experience with other libraries in this field. These libraries very often feature an object-oriented structure that makes it very hard to trace errors or to extend or customize the library due to its complexity. The new component model library offers a unique structure that is simple to understand and that fulfills the requirements of developers, simulation specialists, and design engineers. The equations in each component model are formulated to yield a numerically efficient differential algebraic equation (DAE) system. The component model library uses a new fluid property library that offers a generalized and numerically efficient way to include external fluid property computation codes in Modelica and a number of software tools. Both libraries are extended by a team of developers and used to analyze various thermodynamic systems. Two demonstrating examples are presented as a proof of concept for the new model libraries and to show possible applications.

Computer simulations in the field of system simulations allow for a detailed understanding of components and systems as well as for the reliable prediction of system behavior. System simulations also allow for the development and evaluation of control strategies by modeling the control process. Many simulation tools are designed for specific application domains and feature proprietary modeling languages for the formulation of user models. Free equation-based modeling languages such as VHDL-AMS (Ashenden et al., 2002) or Modelica (Tiller, 2001) that are supported by a number of different simulators overcome these limitations. These languages also allow for the coupling of models from different domains, for example the coupling of a system model with an electronic controller in a single simulation. As illustrated in figure 1.1, three types of users can be identified that develop and work with component model libraries written in the modeling languages mentioned above.



Developer **Simulation Specialist** **Design Engineer**

Figure 1.1: Types of users of modern modeling languages at the example of Modelica.

Figure 1.1 uses the example of a Modelica model to demonstrate the different types of users. The left picture shows the source code of the model. The *developer* is used to work in this layer. His major interests on the component model level are to maintain existing models by extending, simplifying, or fixing them and to implement new models. The equations formulated in each component model represent the engineering knowhow of the developer or an entire team of developers and should thus be formulated in a sustainable way with as little overhead as possible. On the component model library level the developer is responsible for the over-all structure as well as for the combination of different models into subsystems and systems. The developer that initially creates a new component model library can handle almost any structural complexity since he knows all important relations and details. But other developers that are interested in the maintenance and extension of an existing component model library also need to understand its object-oriented structure. These developers obviously prefer a structure of the models as well as of the library that is simple to understand and as intuitive as possible.

The *simulation specialist* uses the component models provided by the developer and combines them into system simulations as depicted in the center picture in figure 1.1. His expertise is a good knowledge of system simulations and of numerical solution techniques. The simulation specialist might have to dig into the source code of individual component models to

improve the solution process or sometimes even to obtain a solution. He needs to understand the structure of component models and of the component model library without having to spend too much time on it. Automated visualization of the numerical solution process and of the obtained numerical results are very important for the simulation specialist for a fast and reliable analysis of the developed system models.

The third type of users is the *design engineer*. A design engineer often needs to answer specific questions about the system behavior of existing real-world systems or to evaluate the system behavior of new systems to be built in the future. He uses the system models prepared by the simulation specialist and modifies parameters to analyze the system behavior. The structure of the component model library and of the models is of little concern to him. He instead requires specialized tools that allow for a convenient specification of parameters during preprocessing or for extensive parameter studies. Like the simulation specialist, the design engineer needs tools that allow for a quick visualization of numerical results so that he can evaluate the characteristic properties of the examined systems.

Sustainable component model libraries have to be designed to support all three types of users. Currently available component model libraries in the field of thermodynamic system simulation focus on one or two of the mentioned types of users. The new component model libraries proposed in this thesis take into account the special requirements of all three groups: They feature an object-oriented structure that is simple to understand yet flexible enough for extensions. The equations in the component models such as the conservation laws or the heat transfer and pressure drop correlations are formulated in a numerically efficient and comprehensible way. Two demonstrating examples describing a prototype Peltier water-water heat exchanger and an ejector refrigeration system are provided as a proof of concept. Furthermore, software tools are presented that allow for an automated object-oriented analysis of component model libraries and of component models and for the automated visualization of numerical results during and after the solution process. The new open-source component model libraries are already used and extended by a team of developers at the Institut für Thermodynamik (IfT) and the TLK-Thermo GmbH.

The following section provides a brief overview of the historical development of modeling languages and especially of Modelica. Section 1.3 gives an overview of the state of the art regarding existing component model libraries in the field of thermodynamic simulations. The major objectives of this thesis are summarized in section 1.4. This chapter closes with a presentation of the structure of the thesis at hand.

1.2 A Brief History of Modeling Languages

Figure 1.2 shows a timeline covering the development from the third generation of procedural languages in the 1950s all the way to the release of the Modelica 3.0 specification in 2007. The major goal of modeling languages is to model real-world systems using ordinary differential equations (ODE) that are translated and solved by a simulation tool. For a long time in the history of programming languages, equations were very rare. Even the first version of

Lisp and computer algebra systems focused more on formula manipulation than on direct simulation (Fritzson, 2004).

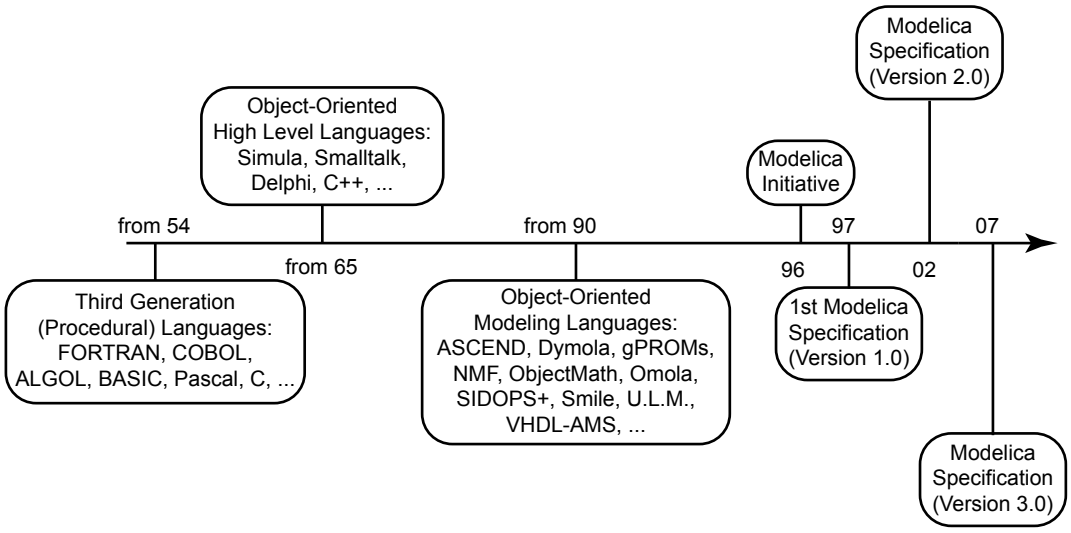


Figure 1.2: Timeline of the Modelica development.

Simula I is a superset of Algol 60 introduced by Dahl and Nygaard (1966). It quickly got a reputation as a modeling language but also served as a general purpose programming language. Simula I was displaced by Simula 67 that took full advantage of the newly invented inheritance mechanism. Simula 67 introduces objects, classes, subclasses, virtual methods, coroutines, and discrete event simulation and could be specialized for many domains including system simulations. Simula was used as a platform for the development of Smalltalk that was initially released in the 1970s. Smalltalk extends object-oriented programming by the integration of graphical user interfaces and interactive program execution. The development of C++ that started in 1979 brought the key concepts of Simula in the C programming language trying to overcome the main disadvantage of Simula which is that is was a rather slow language (Nance, 1995).

Elmqvist (1978) presented a new modeling language in his PhD thesis that he called Dymola. This stands for Dynamic Modeling Language and is not to be mixed up with today's meaning of Dymola which is Dynamic Modeling Laboratory. Elmqvist was the first to describe the importance of modeling with acausal equations using hierarchical submodels and methods for automatic symbolic manipulation to support the solution of equations (Fritzson, 2004). The Dymola language was enhanced in 1993 by including inheritance, a mechanism for discrete-event handling, and more efficient solution methods. Other modeling languages at the same time introduced important new features. The GASP-IV system described by Pritsker and Hurst (1973) is a FORTRAN-based modeling language introduced in 1974 and replaced by the GASP-V system a few years later that introduced integrated continuous-discrete simulation. The Omola language that was first released in 1989 is another equation-based and object-oriented modeling language that supports hybrid simulation (see Mattsson and Andersson, 1992). Other early object-oriented modeling languages are NMF

(Neutral Modeling Format) introduced in 1989 and primarily used for building simulation, Allan-U.M.L., ASCEND, gPROMS, SIDOPS+ supporting bond graph modeling, and SMILE initially released in 1995 and influenced by Objective C. ObjectMath is an object-oriented computer algebra and simulation system integrated with Mathematica that was initially developed by Peter and Dag Fritzson starting in 1989 (Viklund and Fritzson, 1995).

This fairly brief survey of the history of object-oriented modeling languages already shows one major problem that simulation specialists faced in the late 1990s: The shear number of modeling languages made it hard to decide which language to use for solving a specific problem. Against this background, a group of tool designers, application experts, and computer scientists joined forces in September 1996 within the scope of the ESPRIT project Simulation in Europe Basic Research Working Group (SiE-WG). This group brought together the specialists behind Dymola, Omola, ObjectMath, NMF, Allan-U.M.L., SIDOPS+, and Smile (Elmqvist and Mattson, 1997). The major goal of this group was to write a white paper on object-oriented modeling language technology in which the possible unification of existing modeling languages should be discussed besides more general topics. However, the group quickly moved ahead from just proposing a possible unification of existing languages and moved to the more ambitious goal of creating a new modeling language from scratch. The new modeling language was called *Modelica*. The first Modelica language description was published on the web in September 1997. The unique feature of Modelica is the combination of an equation-based object-oriented modeling language with a technology for graphical editing that allows for the design of model libraries with predefined component models that can be used graphically to assemble systems. The small group of developers of Modelica managed to find support within EuroSim and within the Society for Computer Simulation International (SCS). In February 2000, the Modelica Association was founded as an independent nonprofit organization to support the further development of the Modelica language and the Modelica Standard Library. The Modelica language itself allows the formulation of dynamic model properties in a declarative way through equations. The newest language specification is Modelica 3.0 which allows for the simulation of systems containing millions of equations which has practically not been possible with the Modelica 2.x language features. The Modelica Association is formed by people from the industry as well as academic researchers covering a wide range of different educational backgrounds. The diversity of the Modelica Association is one of its biggest advantages. The object-oriented features of Modelica are explained in chapter 2.

1.3 State of the Art in Thermodynamic System Simulations

The simulation of thermodynamic systems such as refrigeration, air-conditioning, and heat-pump systems has been the focus of many researchers over the last decades. A number of publications using various modeling languages and tools have been presented regarding this topic. This section describes the state of the art for component model libraries for thermodynamic systems. Other good introductions to this field including comprehensive overviews can be found in the theses of Tegethoff (1999) and Pfafferott (2004).

Component model libraries for the simulation of thermodynamic systems exist in different forms. One form is the bundling of a closed-source component model library with a simulation tool into a single product. Examples for such bundled products are Sinda/Fluint (Cullimore and Hendricks, 2001), Flowmaster2 (Burke and Haws, 2001), or AMESim (Roccatello et al., 2007). These software tools usually focus on simulation specialists and design engineers by providing ready-to-use component models that can be parameterized but not changed by the user. The development of new user-defined component models is usually supported but not the design of completely new component model libraries. These software tools are thus not applicable for the design of open-source component model libraries.

Other component model libraries are formulated in proprietary modeling languages and can thus only be used with the according software tool. Becker (1996) and Hasse et al. (1996) use the program system MATLAB/Simulink for the dynamic simulation of cold-storage plants. The characteristics of MATLAB/Simulink require special user interventions to achieve numerical convergency. Another tool with a proprietary modeling language is the Engineering Equation Solver (EES) developed by Klein (1999) for the steady-state and transient simulation of thermodynamic systems. It supports non-linear algebraic systems of equations and provides comprehensive fluid properties. The simulation tool CoolPack described by Andersen et al. (1999) for the simulation of refrigeration systems was developed using EES. Both presented languages do not support the creation of object-oriented component model libraries.

Another form of component model libraries are simulation platforms written in common programming languages such as FORTRAN or C++. These simulation platforms usually contain a number of component models and provide a framework for new user-defined models. Examples for such simulation platforms are the GPA program system used by Adiprasito (1998), the simulation platform in C++ developed by Tegethoff et al. (2004) that is described in detail beneath, or the simulation software tool ACCO2 originally developed in FORTRAN (Robinson and Groll, 2000) and then converted to C++ (Ortiz et al., 2003). All of these simulation platforms use a procedural approach to formulate the describing equations within each component model. The major drawback of this approach is that the possible interconnections of components are limited due to the strict interface definitions that are required for each component model. This limitation is overcome by equation-based modeling languages presented in the following paragraph.

The most promising starting point for the development of an open-source component model library are equation-based modeling languages such as VHDL-AMS (Ashenden et al., 2002) or Modelica (Tiller, 2001) that are supported by a number of different simulators. Simulators for VHDL-AMS are for example Advance MS, Smash, or Simplorer and simulators for Modelica are for example Dymola, MathModelica, or SimulationX. VHDL-AMS is a derivative of the hardware description language (HDL) described in the IEEE standard 1073-1993 and includes analog and mixed-signal extensions (AMS) in order to define the behavior of analog and mixed-signal systems according to IEEE 1076.1-1999 (Christen and Bakalar, 1999). VHDL-AMS is almost exclusively used for the modeling of analog, digital, and mixed analog/digital circuits and is an industry standard modeling language. Although it can also

be used to model heterogeneous real-world systems (see for example Haase et al., 2004), only few component model libraries with very basic component models for the simulation of thermodynamic systems are available. This is different for the relatively new object-oriented equation-based modeling language Modelica (see also section 1.2). Many different component model libraries are available in Modelica and have been presented at the five International Modelica Conferences in 2000, 2002, 2003, 2005, and 2006[1], at other scientific conferences, and in a number of journal articles (for example Otter et al., 1999-2000). The following paragraphs provide information on the most advanced Modelica libraries for thermodynamic applications.

Tummescheit, Eborn, and Wagner developed the `ThermoFlow` library that is a Modelica base library for thermo-hydraulic systems described by Tummescheit et al. (2000). A more detailed description including a couple of demonstrating examples can be found in Eborn (2001). Eborn (2001) also discusses fundamental aspects regarding the design of model libraries. The `ThermoFlow` library uses multiple inheritance (see section 2.3.3) as the main structuring method. Further applications of the library that was eventually renamed to `ThermoFluid` are described in Tummescheit and Eborn (2002). Tummescheit (2002) also describes general aspects regarding the design and implementation of object-oriented model libraries. The `ThermoFluid` library was developed with the goal to design and implement a well validated, easy-to-use model library for the dynamic simulation of thermo-hydraulic processes. Pfafferott (2004) uses the models of the `ThermoFluid` library as a starting point for the development of additional component models to simulate mobile CO_2 air-conditioning systems. He gives a detailed description of the fundamental physical models and uses measurement results to validate the component models. Based on the `ThermoFluid` library, a new Modelica library called `ACLib` was developed in a joint research project by DaimlerChrysler AG, Airbus Deutschland GmbH, and the TU Hamburg-Harburg to simulate mobile R134a and CO_2 air-conditioning systems (see Limperich et al., 2005). Parts of this new `ACLib` were used in the design process of the `AirConditioning` library developed by the Swedish company Modelon AB (formerly Scynamics HB) and presented by Tummescheit et al. (2005a). A more detailed description of the library including advanced compressor models initially developed by Försterling (2003) is given by Tummescheit et al. (2005b). The `AirConditioning` library is the currently most widely used Modelica library to simulate air-conditioning systems. It is used by the *Arbeitskreis Kälte- und Kreislaufsimulation* of the German OEMs with participation of Audi, BMW, Daimler, and Volkswagen for unified steady-state and transient simulation of mobile air-conditioning systems. The structure of the `AirConditioning` library is rather complex indicating that this library focuses on design engineers and partly on simulation specialists but not on new developers (see also section 2.7).

Within the scope of simulating thermo-fluid flow, the `Modelica_Fluid` library has been developed over the past years. This free Modelica library provides component models describing one-dimensional thermo-fluid flow in networks of pipes. The library is developed and maintained by the Fluid Task Force that is part of the Modelica Association and was presented on two Modelica conferences (Elmqvist et al., 2003; Casella et al., 2006). The library

[1] See http://www.modelica.org for more information.

uses the fluid property library `Modelica.Media` which is part of the Modelica Standard Library to model fluid properties. This fluid property library tries to combine all possible fluid models from ideal gases to two-phase fluids into a single framework to allow for an easy model exchange. The resulting object-oriented structure is rather complex and the library is hard to understand for new developers and users even with a good knowledge of object-orientation (see also section 2.7). The development of the `Modelica_Fluid` library itself has proven to be very difficult due to the different perceptions of the participating developers regarding fluid flow simulations. Especially the connector design and the formulation of the conservation laws have been a constant source of disagreement. Parts of the ongoing discussion can be found in the minutes of the Modelica Design Meetings. As of the time of writing, no final release of the library has been made, but chances are that there will be an improved version available after the next International Modelica Conference 2008 in Bielefeld.

Some of the basic ideas of the `Modelica_Fluid` library such as the `semiLinear()` function (see section 4.2) have been used in the design process of the open-source component model library developed within the scope of this thesis. Another important starting point for the new component model library was the object-oriented simulation platform for refrigeration, air-conditioning, and heat-pump systems described by Tegethoff (1999).

Tegethoff (1999) provides an in-depth analysis of object-oriented techniques and a detailed motivation for their application. The modeling of components and systems is described based on the object-oriented description of the structure of the library. Tegethoff describes an approach for the formulation of the describing equations in each component model that marks a transition between a purely algorithmic formulation and an equation-based approach. The component model equations are transformed manually into an algorithmic formulation using specialist expertise regarding the residues and the algebraic variables. Another focus of his thesis is the determination and description of the air-side heat transfer and pressure drop correlation for fin-and-tube heat exchangers. The simulation platform is implemented in C++. The simulation results are validated with measurement results from bus air-conditioning systems with the refrigerants R134a and CO_2. Lemke (2005) describes two-stage liquid chillers using the refrigerant CO_2. He uses the simulation platform described by Tegethoff (1999) for steady-state simulations and Modelica models for transient simulations. The dynamic models use a specific formulation of the momentum balance first presented by Tegethoff et al. (2004) which assumes the time derivative of pressure to be locally constant at each pressure level. Lemke uses the results from a test facility to validate the numerical results.

The analysis of the state of the art in the simulation of thermodynamic systems showed the need for a new approach to structure component model libraries to obtain a sustainable library structure that is simple to understand and powerful to allow for various extensions.

1.4 Objectives

The main objective of this thesis is the analysis of general strategies for the development of object-oriented component model libraries as well as the proposal of suitable basic structures. The applicability of the proposed strategies and structures shall be demonstrated in a

new object-based fluid property library and a component model library for thermodynamic systems to be developed in the equation-based modeling language Modelica.

In order to allow for an object-oriented analysis of Modelica component library graphical representations for all object-oriented features of the modeling language shall be formally introduced. Using these graphical representations a tool for the automated generation of class diagrams shall be developed to allow for an easy evaluation of the object-oriented structure throughout the entire design process.

The new fluid property library shall be based on a generalized approach to include external fluid property computation codes in Modelica. It shall be simple to extend to cover additional fluid property computation codes and shall be numerically efficient to be compatible with Modelica-internal solutions.

The design of the new model library for components and systems shall be based on the developed object-oriented analysis. All balance equations shall be formulated in an easy and comprehensible way in base components. All component models shall be easy to understand and to use to improve the process of development, maintenance, and application on all three described levels.

Furthermore, a software tool for an automated online visualization of the solution process and of numerical results in relevant diagrams such as pressure-enthalpy diagrams shall be developed to allow for a quick and reliable evaluation of numerical results and for the detection of modeling errors or numerical difficulties during the solution process.

Applications covering different fields of thermodynamic systems shall be presented to demonstrate the capabilities of the two new component model libraries. The selected applications shall furthermore show the extendibility and multidisciplinarity of the proposed approach.

The following section describes the structure of this thesis that follows in large parts the ordering of the objectives presented above.

1.5 Structure of the Thesis

Chapter 2 describes the fundamentals of an object-oriented analysis based on two simple relations: The *is-a*-relation and the *part-of*-relation that are used to describe the structure of the new component model libraries. It also introduces graphical representations for all object-oriented relations in Modelica based on the UML standard. Simple examples and Modelica code listings are provided to illustrate the different relations. A set of design rules for a good design of object-oriented model libraries are formulated and some existing Modelica libraries are analyzed using the proposed rules.

Chapter 3 explains the different approaches to model fluid properties currently available in Modelica. It extends these approaches to allow for a standardized inclusion of external fluid property computation codes. Based on the analysis of the currently available approaches, a new object-based approach to model fluid properties is presented that is then used throughout the entire rest of the thesis.

The new object-oriented component model library to analyze thermodynamic systems is presented in chapter 4. This chapter focuses on the basics of the new library describing the formulation of the conservation laws, selected component models, as well as the object-oriented structure of complex component models such as the heat exchangers. The last section of this chapter discusses some important numerical aspects.

Chapter 5 describes visualization techniques that allow for an automated analysis of the object-oriented structure of Modelica libraries as well as for the thermodynamic interpretation of the solution process and of the numerical results.

Chapters 6 and 7 present two demonstrating examples extending the new component model library to allow for the simulation of a prototype Peltier water-water heat exchanger and an ejector refrigeration system.

The last chapter closes with a conclusion summarizing the obtained results and provides recommendations for future work.

Chapter 2

Object-Oriented Modeling

A good knowledge of the basic principles of object-oriented modeling is the key to designing component model libraries that can be developed and used by a team of developers and simulation specialists. This chapter formally introduces graphical representations for all object-oriented features of Modelica based on the UML standard as a standard to be used in future Modelica publications. The major design rules for the new component model library are formulated based on a thorough object-oriented analysis. The methods and formalisms introduced in this chapter are used throughout the rest of the thesis.

2.1 Introduction

The overall objective of this thesis is the proposal of a component model library with an object-oriented structure that is simple to understand for students, developers, and simulation specialists and that allows for simulating steady-state and transient thermodynamic systems. To achieve this goal, a thorough object-oriented analysis of the library structure has to be the starting point for the development as well as a benchmark throughout the entire design process. This chapter describes a simple object-oriented analysis based on two relations: The *part-of*-relation representing composition or aggregation and the *is-a*-relation representing inheritance. The result of this object-oriented analysis is a class diagram which is a graphical representation of the analyzed problem. Different formats exist for this graphical representation. The most widely used format is the graphical notation defined in the Unified Modeling Language (UML), a standardized specification language for object modeling (see Fowler, 2003). Many Modelica-related publications such as Eborn (2001), Tummescheit (2002), or Fritzson (2004) use graphical notations from the UML without introducing them formally. The results are minor differences in the graphical representations as well as an incomplete set of elements due to the fact that object-oriented Modelica features like replaceable object types and replaceable local classes are not covered. This chapter makes up for this shortfall by formally introducing UML-style notations for all object-oriented Modelica features. Short Modelica code examples are provided for each language feature to demonstrate

its implementation. The presented graphical representations ensure a consistent graphical description of Modelica problems.

Object-oriented modeling originated with Simula, an Algol-based language that supported discrete simulation as presented in section 1.2 (Booch, 1995). Object-oriented modeling is based on objects with well-defined properties such as dimensions or transfer behaviors. Each object represents a fragment of reality and the combination of multiple objects allows for the construction of complex systems. Modelica uses object-orientation primarily as a structuring concept. Other concepts known from object-oriented programming languages such as dynamic object creation/deletion and dynamic message passing are not part of the Modelica language. The most commonly mentioned advantages of object-oriented programming compared to procedural programming are:

- reusability of the developed source code
- good extensibility and readability of the developed source code
- possibility to develop large libraries
- good feasibility for team work
- polymorphic integration of components

Many scientific publications address the advantages and disadvantages of object-oriented programming versus procedural programming with differing results. A very good introduction to object-oriented design including a detailed discussion of all presented features is given by Eckel (2006).

This chapter is structured in the following way: Classes and the fundamental object-oriented relations are explained in sections 2.2 and 2.3 respectively. Polymorphism that is used to make types and classes exchangeable in Modelica is explained in section 2.4. Graphical notations for polymorphic relations based on the UML standard are introduced to allow for a consistent representation of these relations in class diagrams. Some of the potential pitfalls in object-oriented modeling are discussed in section 2.5.

Accessing object attributes is very important in Modelica especially when combining multiple objects to a system. A complete overview of different techniques to access object attributes in Modelica including examples is given in section 2.6. Section 2.7 evaluates some existing Modelica libraries based on the formulated design rules to point out possible disadvantages if the proposed design rules are violated. A short overview of all design rules is given in appendix B.

Like many other programming languages, Modelica defines the data types `Boolean`, `Integer`, `Real`, `String`, and `Enumeration`. These data types can be further specialized using keywords such as `parameter` or `constant`. The full set of keywords and further details can be found in the language definition itself (Modelica Association, 2005) and in secondary Modelica literature such as Fritzson (2004).

2.2 Classes

Classes are the basic elements of any object-oriented analysis. A class is the abstraction of a number of objects with similar properties. It has to be able to represent all important features of the objects it is representing. Objects are created based on this abstraction and are called instances or - if they are part of a system - components. It is common practice that class names begin with an uppercase letter and object names with a lowercase letter to easily discriminate between both of them. This is also stated in the first design rule. Note that there are some common exceptions to this rule such as writing `T` for a temperature. It is furthermore common practice in Modelica to include the path when referencing class names.

Design Rule 1:	Class names should begin with an uppercase letter whereas object names should begin with a lowercase letter. Exceptions to this rule are allowed for object names if they refer to common abbreviations marked by an uppercase letter and if chances are low to mistake them as class names.

There are different notations to present classes in class diagrams (OMG, 2007; Martin and Odell, 1998). The Unified Modeling Language (UML) developed by the Object Management Group is a standardized object-oriented language to model software. The UML notation is a de facto standard and is used throughout this thesis to visualize classes and relations between classes. A class is marked by a rectangle with several sections as shown in figure 2.1. Most commonly, separate sections are used for the class name, the attributes of the class, and its methods.

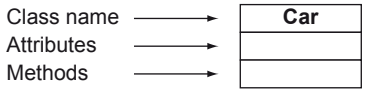


Figure 2.1: Illustration of a class in UML notation.

In Modelica all dynamic model properties are expressed in a declarative way through equations (Fritzson, 2004). Methods known from object-oriented programming languages like Java or C++ are not defined in Modelica. Attributes in Modelica classes can have two different levels of visibility, `public` or `protected`. Only attributes defined as `public` can be read and updated from code that has access to an instance of a Modelica class. Attributes defined as `protected` can only be accessed from code inside the class.

2.3 Object-Oriented Relations

A number of elements exist in the UML to model relations between classes. As far as Modelica is concerned only composition, aggregation, and inheritance are of importance. These three relations are explained in this section. Other relations such as multiplicity or realization also covered by the UML are not defined in Modelica.

2.3.1 Composition

Composition is a technique to combine several objects to form a new system. The objects are becoming components of the new system and the system itself is a container class. Figure 2.2 shows a simple example for a composition, which can also be called a *part-of*-relation: Four tires and an engine are part of a car.

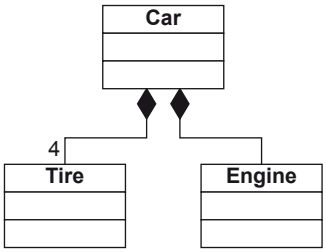


Figure 2.2: Visualization of composition in UML notation.

Code listing 2.1 shows the Modelica code for the composition shown in figure 2.2. An engine and a vector with four instances of type `Tire` are part of the model `Car`.

```
model Tire "Tire model"
end Tire;

model Engine "Engine model"
end Engine;

model Car "Car model"
  Tire[4] tire "Tires";
  Engine engine "Engine";
end Car;
```

Code Listing 2.1: A simple example for composition in Modelica.

2.3.2 Aggregation

Aggregation differs from ordinary composition by not implying ownership. Composition is usually implemented in a way that an object contains another object as shown in code listing 2.1. In aggregation, the object may only contain a reference or a pointer to the other object. The `inner`/`outer`-concept in Modelica can be seen as aggregation although the term aggregation has not yet been used for this concept in any Modelica-related publication. Any composed object in Modelica can be marked with the prefix `inner`. This allows for referencing this object further down in the instance hierarchy using the prefix `outer`. An object specified with the prefix `outer` is thus only a reference to another object. Note that the `inner`/`outer`-concept in Modelica can also be used for referencing type definitions which goes beyond the scope of this section. Refer to the Modelica language specification (Modelica Association, 2005) for further details.

Figure 2.3 shows an example for aggregation in Modelica: The object `RoadConditions` models the road conditions that are important for the tire model but that are not owned by the model `Tire` but by the `VehicleTestBench` in this simple example.

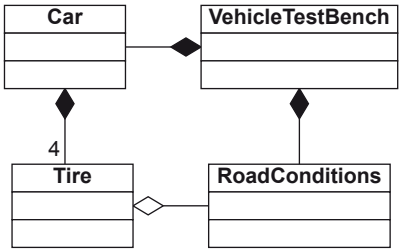


Figure 2.3: Visualization of aggregation in UML notation.

Code listing 2.2 shows the Modelica code for the example presented in figure 2.3. An object declared with the prefix `outer` references the object with the same name but using the prefix `inner` which is nearest in the enclosing instance hierarchy of the outer element declaration. The object `roadConditions` in `Tire` therefore references the object instantiated in `VehicleTestBench`.

```
1  model RoadConditions "Road conditions model"
2  end RoadConditions;
3
4  model Tire "Tire model"
5    outer RoadConditions roadConditions "Road conditions";
6  end Tire;
7
8  model Car "Car model"
9    Tire[4] tire "Tires";
10 end Car;
11
12 model VehicleTestBench "Vehicle test bench model"
13   inner RoadConditions roadConditions "Road conditions";
14   Car car "Car";
15 end VehicleTestBench;
```

Code Listing 2.2: A simple example for aggregation in Modelica.

Using the prefix `inner` for `roadConditions` in line 13 of code listing 2.2 can also be seen as a widening of the scope of this object since it can now be referenced in other objects further down in the instance hierarchy. A further discussion of the `inner`/`outer`-concept can be found in section 2.6.

2.3.3 Inheritance

The third important relationship between classes in Modelica is inheritance. Inheritance in object-orientation means that *all* attributes and methods are passed from the parent to the child class. Inheritance can be regarded as an *is-a*-relation. It is very important to understand that inheritance in object-orientation is not the same as heredity which is the transfer of *some*

characteristics from parent to offspring through their genes. This is even more confusing in German since the same word, *Vererbung*, is used in both contexts. A simple example for inheritance is shown in figure 2.4: A car and an airplane are both a means of transportation.

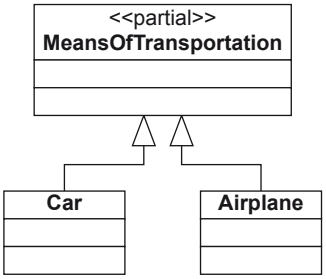


Figure 2.4: Visualization of inheritance in UML notation.

Inheritance can be achieved in two different ways in Modelica. The first way is using the keyword `extends` as shown in code listing 2.3. The keyword `partial` indicates that the model `MeansOfTransportation` cannot be instantiated but has to be inherited before instantiation.

```
partial model MeansOfTransportation "Partial base model for means of transportation"
end MeansOfTransportation;

model Car "Car model"
  extends MeansOfTransportation;
end Car;

model Airplane "Airplane model"
  extends MeansOfTransportation;
end Airplane;
```

Code Listing 2.3: A simple example for inheritance in Modelica.

The second way to achieve inheritance in Modelica is based on the type system of Modelica itself. Modelica uses a type system inspired by Abadi and Cardelli (1996) (see also Pierce, 2002) which is a structural type system as opposed to nominal type systems. In this type system, two classes are type equivalent if they possess the same public attributes. Code listing 2.4 shows a very simple example for type equivalency using the Cardelli type system. It should be noted that not only the types but also the names of the attribute have to be identical.

```
model Person
  String firstName "First name";
  String lastName "Family name";
end Person;
```

```
model Student
  String firstName "First name";
  String lastName "Family name";
end Student;
```

Code Listing 2.4: Type equivalency in the Cardelli type system.

Inheritance or subtyping is achieved in the Cardelli type system without using an explicit keyword. A class that possesses all public attributes of another class and additional public attributes is a subtype of the other class. Code listing 2.5 shows a very simple example of subtyping using the Cardelli type system.

```
model Person
  String firstName "First name";
  String lastName "Family name";
end Person;
```

```
model Student
  String firstName "First name";
  String lastName "Family name";
  Integer studentID "Student ID number";
end Student;
```

Code Listing 2.5: Subtyping in the Cardelli type system.

Throughout the entire thesis, inheritance is treated as an *is-a*-relation which is not the case in many books about object-oriented programming. Figure 2.5 shows an example for inheritance according to Fritzson (2004).

Figure 2.5: UML class diagram for usage of inheritance. The left example illustrates a common usage of inheritance that is discouraged by design rule 2.

The `VerticalLine` in the left class diagram inherits the coordinates from `Point` and adds a length. This cannot be expressed using an *is-a*-relation. The right class diagram shows an alternative version using composition. The `VerticalLine` has a `Point` for example named `startingPoint` and adds a length. The inheritance example from figure 2.5 yields an object-oriented structure that is less intuitive since it cannot be described using an *is-a*-relation. This motivates the second design rule for component model libraries.

Design Rule 2:	Inheritance should only be used if the relation between two classes can be described as an *is-a*-relation.

The Modelica Standard Library uses inheritance in many places to copy graphical information into a class as shown in code listing 2.6. Again the statement *an Add3-block is a block icon* is plainly wrong and the usage of inheritance is discouraged by design rule 2.

```
block Add3 "Output the sum of the three inputs"
  extends Interfaces.BlockIcon;
  ... // further code omitted
end Add3;
```

Code Listing 2.6: Example for inheritance used in the Modelica Standard Library to inherit graphical information.

Inheritance is required for polymorphism that is one of the main features of object-oriented programming. Polymorphism allows for exchanging the type of an object during instantiation as described in section 2.4.

2.3.4 Multiple Inheritance

Modelica also supports multiple inheritance. This means that a class can be derived from more than one base class. Figure 2.6 illustrates an example for multiple inheritance. The Modelica code for this example is shown in code listing 2.7. The class `ChristophRichter` extends from `Student` as well as from `Person`.

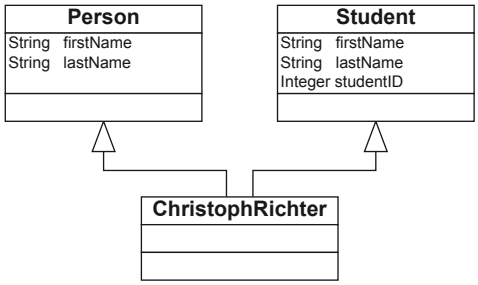


Figure 2.6: UML class diagram for multiple inheritance example.

In order to understand how multiple inheritance works in Modelica, it is important to note that inheritance in Modelica differs from inheritance known from other object-oriented programming languages. In Modelica the data and behavior of the base class such as attribute declarations and equations are copied to the derived class during inheritance. This seems to yield a problem when an attribute in a base class has the same name as a local attribute in the derived class. However two attributes with the same name are legal Modelica if the two attribute declarations are identical. The two attributes are merged into a single declaration. Note that the example presented in code listing 2.7 is only legal because the modifications of `firstName` and `lastName` are identical.

```
model Person
  String firstName "First name";
  String lastName "Family name";
end Person;

model Student
  String firstName "First name";
  String lastName "Family name";
  Integer studentID "Student ID number";
end Student;
```

```
model ChristophRichter
  extends Person(
    firstName="Christoph",
    lastName="Richter");
  extends Student(
    firstName="Christoph",
    lastName="Richter",
    studentID=2614289);
end ChristophRichter;
```

Code Listing 2.7: Multiple inheritance example in Modelica.

Equations are copied from the parent class to the child class during inheritance but are not merged even if they are mathematically identical. This yields an over-determined system of equations in many cases.

Multiple inheritance has been discussed extensively since the introduction of object-orientation. Some object-oriented programming languages such as C++, Perl, or Python support unlimited multiple inheritance. Other languages like Java or C# allow multiple inheritance only for abstract interfaces. Multiple inheritance can easily lead to ambiguities

often summarized as the diamond problem. Figure 2.7 shows a diamond problem taken from Fritzson (2004).

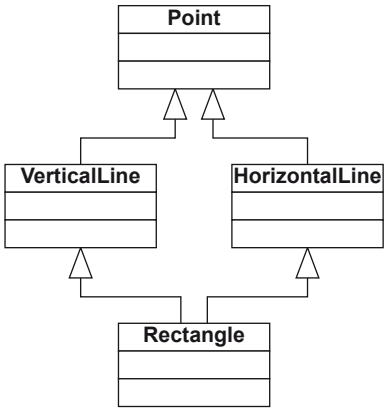


Figure 2.7: Multiple inheritance example from Fritzson (2004).

Note that the example for multiple inheritance shown in figure 2.7 is discouraged by design rule 2 since it does not use inheritance for an *is-a*-relation. This is explicitly formulated in the third design rule.

Design Rule 3:	Multiple inheritance should only be used if each inheritance relation fulfills design rule 2.

Furthermore it should be noted that the presented example for diamond inheritance adds unnecessary complexity to the development process. In general the presented example can be implemented in Modelica without any problems. The `Point` specifies the coordinates and the `VericalLine` and the `HorizontalLine` add information about `height` and `width` of the `Rectangle`. The coordinates defined in `Point` are merged into a single declaration in `Rectangle`. The developer still has to cope with two problems:

1. Possible declarations or modifications of the coordinates in the classes `VerticalLine` and `HorizontalLine` inherited from `Point` have to be identical in order for Modelica to merge them into a single definition in `Rectangle`.
2. When designing the two classes `VerticalLine` and `HorizontalLine`, it seems natural to introduce an attribute `length` in each class. This leads to a problem when the class `Rectangle` is added in a later development step. The two attributes `length` will then be merged into a single declaration yielding an incorrect model. This problem can be overcome by renaming the two attributes `length` to `height` and `width` respectively. However such a change breaks backwards compatibility which has to be avoided at any cost in component model libraries.

These two problems demonstrate that using multiple inheritance in a component model library can cause major difficulties throughout the development process. This motivates the fourth design rule.

Design Rule 4:	Multiple inheritance should be avoided whenever possible.

Scott Meyers, C++ expert and author of numerous books, phrases this design rule differently in (Meyers, 1997) by saying: "Use multiple inheritance judiciously".

2.4 Polymorphism

Two different kinds of polymorphism defined in Modelica are explained in this section: Exchangeable object types and exchangeable local classes. The third kind allows for using replaceable base classes and is not covered in this section since it is removed in Modelica 3.0 and should thus not be used.

Modelica uses the keyword `replaceable` to indicate an exchangeable class or object type. A constraining type can be specified with the keyword `extends`. The constraining type restricts the type of the class or object replacing another class or object to subtypes of the constraining type. By default the original type is used as constraining type. The actual exchange is performed in a modifier using the keyword `redeclare`.

2.4.1 Exchangeable Object Type

The exchangeable object type allows for making the data type of an instance replaceable. Figure 2.8 shows an example for an exchangeable object type: A garage contains a car that is either a VW or a BMW.

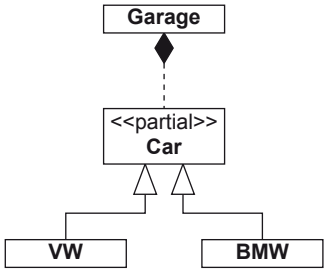


Figure 2.8: Polymorphism in Modelica, exchangeable object type.

Code listing 2.8 shows the corresponding Modelica code. In the class `Garage` a car is instantiated which is a `VW` by default but whose type is made replaceable. It can be replaced using the keyword `redeclare` in the modifier of an instance of `Garage` as shown in line 17.

```
1  partial model Car "Partial car model"
2    String brandName "Brand name";
3  end Car;
4
5  model VW "Model for VW car"
6    extends Car(final brandName="VW");
7  end VW;
8
9  model BMW "Model for BMW car"
10   extends Car(final brandName="BMW");
11 end BMW;
12
```

```
13 model Garage "Garage model"
14   replaceable VW car extends Car "Car in garage";
15 end Garage;
16
17 model House "House model"
18   Garage garage(redeclare BMW car);
19 end House;
```

Code Listing 2.8: Polymorphism in Modelica, code for exchangeable object type.

2.4.2 Replaceable Local Class

The replaceable local class is the second important kind of polymorphism in Modelica. It is very often desired that a number of components in a system have the same - replaceable - type (e.g., tires of a car, resistors in an electrical circuit). This behavior can be achieved by using a replaceable local class as illustrated in figure 2.9.

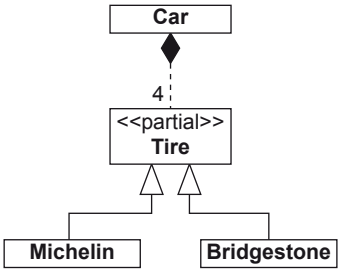


Figure 2.9: Polymorphism in Modelica, exchangeable base class.

Code listing 2.9 shows the corresponding Modelica code. The `Car` contains four tires (lines 15 to 18) which are desired to be from the same brand. This is achieved by defining a new local model `TireModel` in line 14 which is of type Bridgestone by default but can be replaced with all subtypes of `Tire`. The tire model is redeclared in the modifier of the car instance in line 21.

```
1  partial model Tire "Partial tire model"
2    String brandName "Brand name";
3  end Tire;
4
5  model Bridgestone "Model for Bridgestone tire"
6    extends Tire(final brandName="Bridgestone");
7  end Bridgestone;
8
9  model Michelin "Model for Michelin tire"
10   extends Tire(final brandName="Michelin");
11 end Michelin;
12
13 model Car "Car model"
14   replaceable model TireModel = Bridgestone extends Tire "Tire model";
15   TireModel tireFrontLeft "Tire in front left position";
16   TireModel tireFrontRight "Tire in front right position";
```

```
17   TireModel tireRearLeft "Tire in rear left position";
18   TireModel tireRearRight "Tire in rear right position";
19 end Car;
20
21 model TireServiceGarage "Model for a tire−service garage"
22   Car car(redeclare model TireModel = Michelin);
23 end TireServiceGarage;
```

Code Listing 2.9: Polymorphism in Modelica, code for exchangeable base class.

The choice of the keyword `extends` to specify the constraining type was not a good choice since the same keyword is used for inheritance. The Modelica Association decided to change this important keyword to `constrainedby` in the Modelica 3.0 specification (Modelica Association, 2007). Line 14 from code listing 2.9 becomes

```
14   replaceable model TireModel = Bridgestone constrainedby Tire;
```

2.5 More Pitfalls in Object-Oriented Modeling

Some potential pitfalls in object-oriented modeling were presented in the previous sections and led to the formulation of the first four design rules. Some additional potential pitfalls in object-oriented modeling are described in this section.

A common potential pitfall in object-oriented modeling is to design a library without going through an object-oriented analysis beforehand. As described in the previous sections, object-oriented modeling uses objects and their interactions to design complex models of real-world problems. This requires the modeler to break down real-world problems into a suitable class structure to design a model library (Fowler, 1996). This process is never straightforward and its outcome depends on the perspective of the modeler. A simple example is used to demonstrate the benefits of a thorough object-oriented analysis. Assume that a component model library for means of transportation is to be developed from the following limited set of models: `MeansOfTransportation`, `Airplane`, `Ship`, `Car`, `Bicycle`, `Engine`. Using these models, a class diagram as shown in figure 2.10 could be developed.

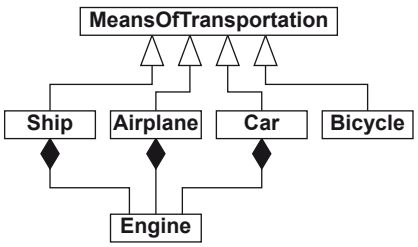


Figure 2.10: UML class diagram to illustrate difficulties with library design.

The class diagram uses the *is-a*-relation and the *part-of*-relation to built up a hierarchical order. The problem that arises is the ambiguity of the presented solution. The class diagram indicates that airplanes have engines but what about gliders? There also are sail boats, kayaks, canoes, and bicycles with electric motors. The structure of the library and of the

component models depends on the problems to be simulated. If muscle powered boats are of interest, a model `MusclePoweredBoat` could be introduced as a specialization of `Ship`. Problems like that are very common and can usually be avoided by performing an object-oriented analysis of the system to model before starting to write the first line of code. It should also be noted that it is very difficult or even impossible to change the structure of a model library at a later stage without breaking user models. This motivates the fifth design rule.

Design Rule 5:	The object-oriented structure of the component model library should be constantly monitored and re-evaluated during the initial design process since it can hardly be changed later on.

An automated software tool to generate class diagrams from Modelica as presented in section 5.2 significantly simplifies this monitoring process.

Another additional potential pitfall in object-oriented modeling are models with many levels of inheritance. Figure 3.1 shows the class diagram of the `ExternalMedia` library that is described in the following chapter. Five levels of inheritance are used to get from `PartialMedium` to `TestMedium` which is very likely to be incomprehensible from a structural point of view for a new user without even going into details regarding the models. Some people argue that this is a tool issue and that a good tool should allow to browse through these different levels without any problems. However a library with a less deep inheritance structure will still be simpler to understand. This is formulated in the sixth design rule.

Design Rule 6:	Component model libraries should feature an inheritance structure that is as flat as possible.

2.6 Accessing Object Attributes

There are four different ways to access public object attributes in Modelica which to the author's knowledge are not clearly listed and compared in any Modelica book or paper. Each of these four ways has its own advantages and disadvantages that have to be taken into account by library developers to find the most suitable solution. The following paragraphs describe each of the four ways and identify their advantages and disadvantages.

2.6.1 Connectors

The most important way to access object attributes in Modelica is to use the specialized class `connector`. Connectors which are often called ports describe the interaction possibilities of a component, i.e., connectors in Modelica are the communication interfaces for communication between a component and the outside world. Examples for connectors are an electrical pin, a mechanical flange, or an input signal. Objects are connected using the `connect()` function that takes two connectors as arguments (see also section 4.2). Most connections in Modelica are acausal which means that the data flow in the connection in not explicitly specified. Connections can also be made causal using the prefixes `input` and `output` in the declaration of at least one of the connectors in the connection specifying the direction of data flow. Note

that there also exists a third type of connections in Modelica called composite connections which are described in detail by Fritzson (2004).

There are two kinds of variables in connectors, namely *nonflow* variables and *flow* variables. Nonflow variables represent some kind of potential or energy level (e.g., voltage, pressure, or position) whereas flow variables represent some kind of flow (e.g., electrical current, force, or fluid flow). The coupling established by the `connect()` function depends on whether the variables in the connectors are nonflow variables (default) or flow variables. For nonflow variables, equality coupling is applied and flow variables are coupled using sum-to-zero coupling according to Kirchhoff's current law. A simple example is provided in code listing 2.10. A connector `Pin` is defined that models an electrical pin. Two instances of this connector called `positivePin` and `negativePin` are created in `Resistor`. The `CircuitBoard` model instantiates two resistors and connects the `negativePin` of the first resistor with the `positivePin` of the second resistor using the `connect()` function.

```
connector Pin "Pin of an electrical component"
  Real v "Potential of the pin";
  flow Real i "Current flowing into the pin";
end Pin;

model Resistor "Ideal electrical resistor"
  Pin positivePin "Positive pin";
  Pin negativePin "Negative pin";
  parameter Real R=1 "Electrical resistance";
equation
  R*positivePin.i = positivePin.v - negativePin.v;
  positivePin.i + negativePin.i = 0;
end Resistor;

model CircuitBoard "Simple circuit board"
  Resistor[2] resistor "Resistors";

  ... // further code omitted
equation
  connect(resistor[1].negativePin, resistor[2].positivePin);
end CircuitBoard;
```

Code Listing 2.10: Simple example for connectors and `connect()` in Modelica.

The described usage of instances of the Modelica class `connector` is sometimes referred to as *explicit* connection structure. The counterpart to an explicit connection structure is an *implicit* connection structure that does not require explicitly formulated `connect()` functions but uses the `inner`/`outer`-concept also discussed in sections 2.3.2 and 2.6.4. Implicit connections are described in detail by Fritzson (2004).

2.6.2 Direct Access

Public object attributes can be directly accessed in the equation section of the container class after instantiation. Code listing 2.11 shows a simple example for a direct access. The

equation for the air pressure of the spare tire is defined in the equation section of the car model. The given example is legal in Modelica 2.x but will be illegal in Modelica 3.0 since the tire model is not locally balanced, i.e., the local number of equations does not equal the local number of unknowns. Direct access is very often an indication of ill defined model boundaries and should be avoided wherever possible.

```
model Tire "Tire model"
  Real airPressure "Air pressure";
end Tire;

model Car "Car model"
  Tire spareTire "Spare tire";
equation
  spareTire.airPressure = 3.2e5 - time;
end Car;
```

Code Listing 2.11: Example for direct access to object attributes.

2.6.3 Modifiers

In Modelica, modifiers can be used to change object attributes during instantiation. Code listing 2.12 shows a simple example for the usage of a modifier. The model `House` contains a parameter to specify the base area with default value of 100 m^2. The model is instantiated in `Kanzlerfeld` which is a district of Braunschweig. The instantiation can be regarded as the creation of an unnamed class used only for the instance (or instances if an array dimension is specified) created by the attribute declaration. This unnamed class is the *building plan* for the created instance. In the example the `House` type of `myHouse` has its default declaration equation for its member `baseArea` replaced by the value specified in the modifier.

```
model House "Model for a house"
  parameter Real baseArea=100 "Base area in m^2";
end House;

model Kanzlerfeld "Model for district of Braunschweig"
  House myHouse(baseArea=200) "My house";
end Kanzlerfeld;
```

Code Listing 2.12: Example for a modifier in Modelica.

Modifiers can not only be used to change parameters as shown in code listing 2.12, but also to redeclare an exchangeable object type as shown in line 17 in code listing 2.8. Again, an unnamed class is created that serves as type for `garage` where the original model `VW` is replaced by `BMW`. The third way to use modifiers is to redeclare an exchangeable local class as shown in line 21 in code listing 2.9.

Besides the presented ways to use modifiers in Modelica, it is also possible to provide additional equations during instantiation. Code listing 2.13 shows an example where the equation for the radio-active decay is not part of the model `RadioActiveSample` but specified in the modifier during instantiation.

```
model RadioActiveSample "Model for radio−active sample"
  parameter Integer n0=100 "Initial number of atomic nuclei";
  parameter Real halfLife=10 "Half−life";
  Real n "Number of atomic nuclei";
end RadioActiveSample;

model Laboratory "Model for laboratory"
  RadioActiveSample sample(n=sample.n0*2^(time/sample.halfLife))
   "Radio−active sample";
end Laboratory;
```

Code Listing 2.13: Example for a modifier in Modelica (to be avoided).

Although allowed in the language specification this usage of the modifier should be avoided at any cost. Supplying equations for variables in the modifier is an indication of ill defined model boundaries and should be completely avoided.

Note that the example from code listing 2.13 will become illegal in Modelica 3.0 because `RadioActiveSample` is not locally balanced and only binding equations are going to be allowed in modifiers. Binding equations are equations for parameters and for variables declared as `input`. The language specification for Modelica 3.0 (Modelica Association, 2007) describes the new restrictions in detail.

2.6.4 `inner/outer`-Concept

The `inner/outer`-concept in Modelica which was explained in section 2.3.2 can also be used to access object attributes. In the given example (see code listing 2.2), the road conditions can be specified by changing the attributes in the inner instance of `RoadConditions` in the vehicle test bench. The changed road conditions are then visible in the tire model in the outer instance of `RoadConditions`. The `inner/outer`-concept is a very elegant way to provide information to models further down in the instance hierarchy. It is very often used for general information that should be accessible from all levels of a hierarchical model such as geometric information or ambient conditions (e.g., ambient pressure and temperature). The new component model library described in detail in chapter 4 uses this concept in a couple of places. The most important ones are:

- To propagate fluid property information from the basic cells to the heat transfer and pressure drop models (see section 4.7)
- To provide geometric data to all levels of the heat exchanger models (see section 4.8)
- To propagate system information (e.g., total refrigerant mass) from components to the system information manager and vice versa (see section 4.10.1)

Note that the `inner/outer` can unnecessarily complicate models if it is used the wrong way. Information that is only relevant for a single component or small set of components within a system should not be propagated using the `inner/outer`-concept but rather by using modifiers or equations to avoid complexity. It is also important to keep in mind that models with `outer` components can only be used when an `inner` component is specified further up in the enclosing instance hierarchy.

2.7 Evaluation of Existing Libraries

This section evaluates particular parts of two existing component model libraries that violate the formulated design rules and demonstrates the consequences. The first example is taken from the fluid property library `Modelica.Media` that is part of the Modelica Standard Library which is described in detail in chapter 3. It uses packages to model the fluid properties of various different fluids such as ideal gases, mixtures, or two-phase fluids. The class diagram of this library is shown in figure 2.11. All 1,240 ideal gas packages extending from `SingleGasNasa` except for `H2O` are skipped for simplicity. The fixed region water packages extending from `WaterIF97_fixedRegion` are also skipped.

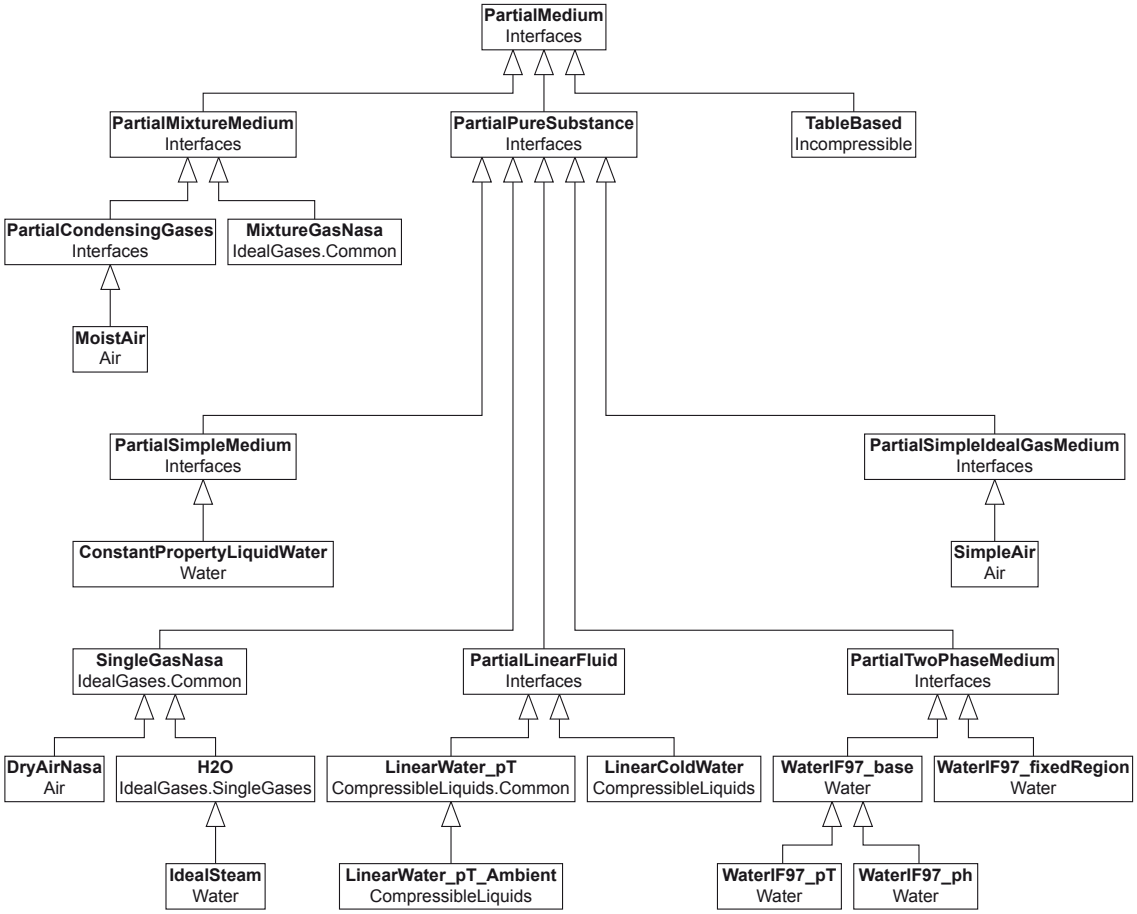


Figure 2.11: UML class diagram of `Modelica.Media` library included in the Modelica Standard Library 2.2.2. Additional 1,240 packages extending from `SingleGasNasa` and all fixed region water packages extending from `WaterIF97_fixedRegion` are skipped for simplicity.

The first thing to notice about the structure of the `Modelica.Media` library is the huge number of classes. Having 1,241 ideal gas packages extending from `SingleGasNasa` just to set some specific names and constants seems to be a flaw in the library design and unnecessarily complicates the resulting structure. The second maybe even more fundamental issue is the number of inheritance levels. The maximum number of inheritance levels in the

`Modelica.Media` library is four (e.g., for `IdealSteam`). Even the inheritance tree for partial base classes such as `PartialTwoPhaseMedium` is two levels deep and yields an even deeper structure if new medium packages extend from these partial base classes. A good example for this is the external fluid property interface developed within the scope of this thesis whose class diagram is shown in figure 3.1. The inheritance tree in this library is five levels deep making it very hard to trace declarations as well as the definition of methods even for developers that developed the library. A third problem with the structure of the `Modelica.Media` library is that the chosen names are not always intuitive. A simple example is considered to demonstrate this problem. The `TableBased` medium sounds like a very simple medium model that uses tabulated values to interpolate fluid properties. But if it is a simple medium model why is it not extending `PartialSimpleMedium`? All three presented points lead to the proposal for a much simpler design for an object-based fluid property library that is described in section 3.5.

The second example is taken from the `ThermoFluidPro` library that is used as a base library in the `AirConditioning` library. The `AirConditioning` library is the most widely used commercial Modelica library to simulate air-conditioning systems as mentioned in section 1.3. One of the basic elements in the `ThermoFluidPro` library is the `HXFluidFlowMB` which is the refrigerant-side base element in all heat exchanger models. The class diagram of this model only visualizing inheritance relations is shown in figure 2.12.

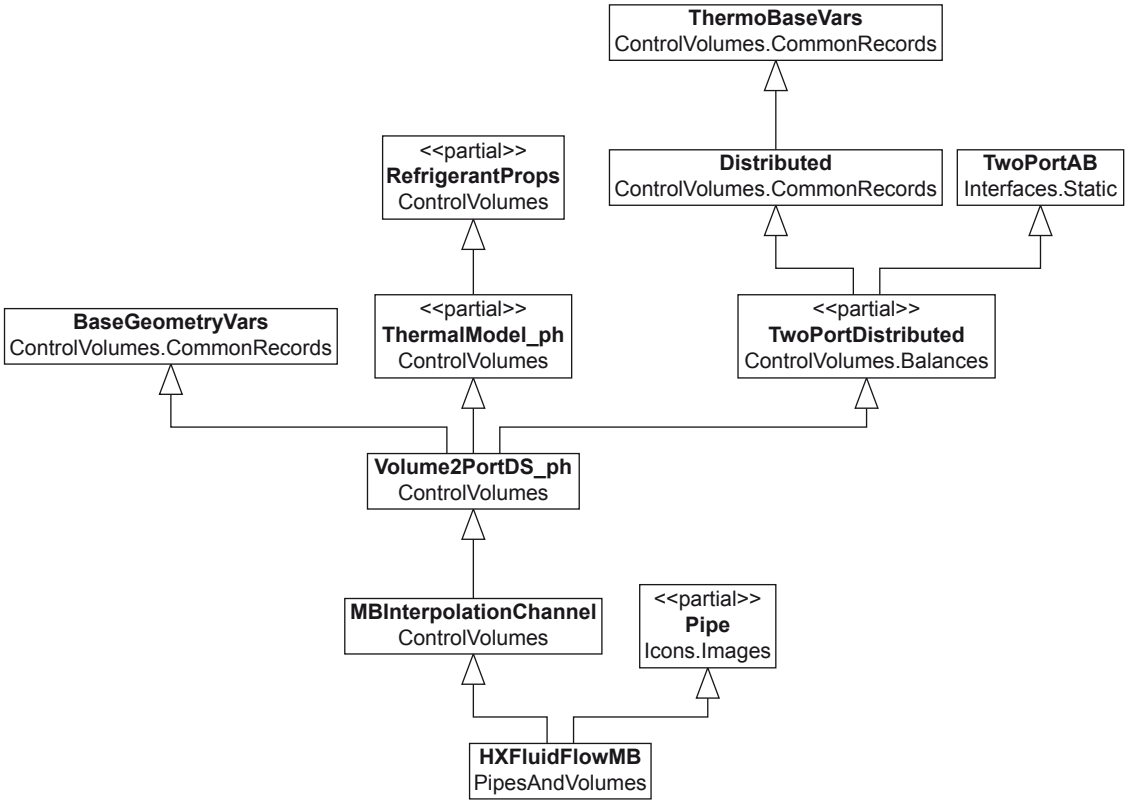


Figure 2.12: UML class diagram of `HXFluidFlowMB`, the refrigerant-side base element of heat exchanger models in the `ThermoFluidPro` library. Only inheritance is shown.

The inheritance tree presented in figure 2.12 is five levels deep and uses multiple inheritance in three places. It should also be noted that the inheritance relations in this diagram cannot be expressed as *is-a*-relations. The basic design concept yielding a complex class structure like that is described for the `ThermoFlow` library, one of the predecessors of the `AirConditioning` library by Eborn (2001). The basic element in the `ThermoFlow` library is the model of a control volume. This model is formed by inheriting from the partial thermal model, the partial hydraulic model, and the medium property model using multiple inheritance. Eborn (2001) states that by using multiple inheritance, "the class tree becomes difficult to visualize and understand since it will become a rather complicated network". Despite this fact, Eborn (2001) still regards the design pattern using multiple inheritance as superior. The `ThermoFlow` library was renamed to `ThermoFluid` library during the development process. Tummescheit (2002) who describes many of the design rules used for the `ThermoFluid` library tries to disprove the general argument against multiple inheritance that it adds more complexity than benefits and states that "there are ... situations when a solution using multiple inheritance is simpler than other alternatives". Multiple inheritance has thus "to be used with care". He also argues that using multiple inheritance ensures a compact design especially when several different implementations of some equations exist and provides examples to confirm this argument. He uses a quote from Abelson et al. (1985) that "programs must be written for people to read, and only incidentally for machines to execute". The class diagram in figure 2.12 shows that this principle is easily violated when using multiple inheritance as explained in a simple example in section 2.3.4. The derived solution for the base element of the refrigerant-side base element of all heat exchanger models is hard to understand for other developers or simulation specialists and implementing new models within this structure or finding modeling errors becomes a time consuming task. The refrigerant-side base element for heat exchangers in the new component model library follows a much simpler design as shown in figure 2.13. The `RefrigerantCell` model is described in more detail in section 4.7.

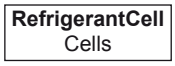


Figure 2.13: UML class diagram of `RefrigerantCell`, the refrigerant-side base element of heat exchanger models in the new component model library presented in chapter 4. No inheritance is used. The complete class diagram of the `RefrigerantCell` is shown in figure 4.13.

A consequence of the simple design of the `RefrigerantCell` is that the conservation laws have to be formulated several times throughout the entire component model library since they are not inherited into the component models using multiple inheritance. Figure 2.14 shows a class diagram of selected component models from the new component model library. The gray component models contain explicit formulations of the conservation laws. The duplication of source code for the formulation of the conservation laws adds a slight overhead but is still regarded as easier to understand and simpler to maintain than a complex object-oriented structure as shown in figure 2.12. Examples for component models and the corresponding

Modelica code that formulates the balance equations are given for the accumulator model in section 4.4, for the refrigerant cell in section 4.7.1, and for the compressor in section 4.9.

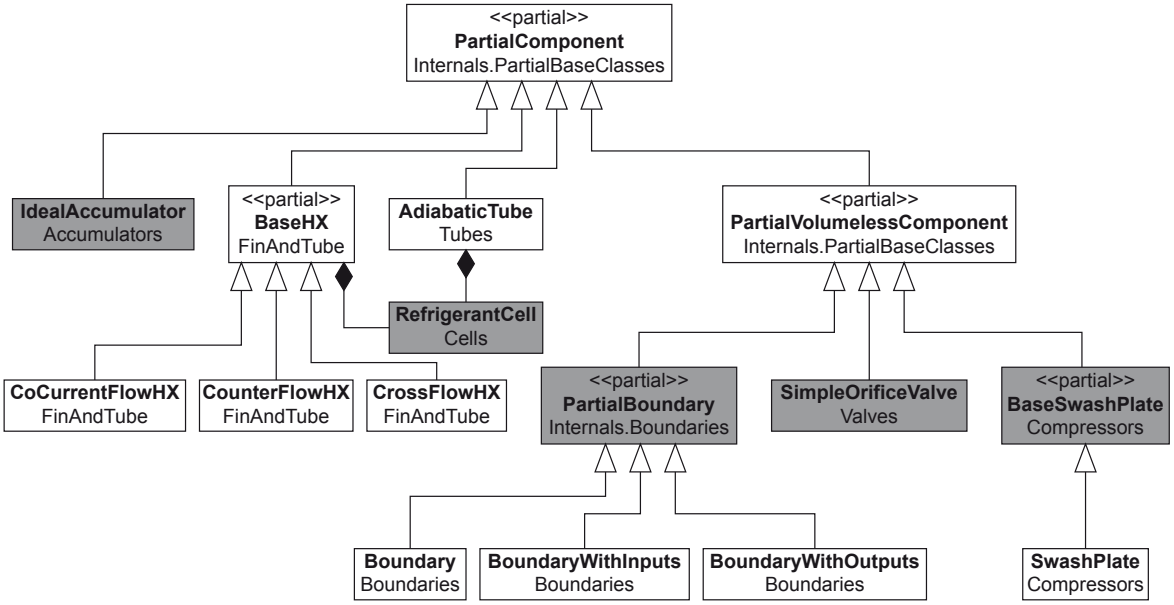


Figure 2.14: Class diagram for selected component models from the new object-oriented component model library. Composition is only shown for the `RefrigerantCell`. The gray classes contain balance equations.

Chapter 3

Modeling of Thermo-Physical Fluid Properties

The modeling of thermo-physical fluid properties has a significant impact on simulations of thermodynamic systems. This chapter presents a new extension to the `Modelica.Media` library based on a generalized approach to include external fluid property computation codes. The presented approach is shown to be compatible with available Modelica-internal solutions. Based on an object-oriented analysis of currently available Modelica libraries to compute fluid properties, a new object-based fluid property library with a significantly simpler structure is presented that is used in all new component models.

3.1 Introduction

The computation of fluid properties is a very important task when simulating refrigeration, air-conditioning, and heat-pump systems and accounts for a significant and sometimes even for the largest part of the required computing time. The design of a fluid property library should thus focus on numerical efficiency as well as on reasonable, application-dependent accuracy. Many existing fluid property libraries are purely function-based which means that they provide functions to compute thermodynamic and transport properties from other known properties. Using function-based fluid property libraries in an object-oriented programming language tends to be cumbersome because of the mixing of two different programming paradigms and does not take advantage of the object-oriented features presented in chapter 2.

The Modelica Standard Library starting with version 2.2 features a fluid property library called `Modelica.Media` that introduces a function-based and an object-based approach to compute fluid properties in Modelica. Elmqvist et al. (2003) and Casella et al. (2006) describe the library and some applications in two papers presented at the International Modelica Conferences in 2003 and 2006 respectively. Modelica packages are used as the basic structuring concept to model fluid properties. Each medium package extends from a partial

base package called `PartialMedium` and implements the equations required to compute all fluid properties. The `Modelica.Media` library contains ready-to-use medium models for ideal gases, mixtures of ideal gases, water/steam, moist air, table-based incompressible fluids, and generic linear fluids. A class diagram of the library is shown in figure 2.11. The function-based approach provided by the `Modelica.Media` library is explained in section 3.2 and the object-based approach and its unique advantages are presented in section 3.3. A major drawback of the library is that all fluid property models have to be implemented in Modelica requiring a time-consuming reimplementation of existing fluid property computation codes. The new Modelica code can then only be used in Modelica models. This drawback can be overcome by providing an interface to external fluid property computation codes that is compatible with the interface defined in the `Modelica.Media` library. The new `Modelica_ExternalMedia` library developed in close cooperation with Prof. Casella from Politecnico di Milano is described in section 3.4. It provides an interface to external fluid property computation codes fully compatible with the `Modelica.Media` library and an interface library written in C++ to handle multiple external fluid property computation codes in an efficient way.

The object-oriented structure of the `Modelica.Media` library is quite complex as discussed briefly in section 2.7 due to its attempt to provide a single interface for different types of fluids such as gases, liquid, and refrigerants. Based on an object-oriented analysis and the design rules presented in chapter 2, a new object-based fluid property library is presented in section 3.5 that features a structure that is much simpler to understand by offering separate medium models for different types of fluids. This differentiation of different fluid models such as gases, liquids, and refrigerants requires an according differentiation regarding the control volumes that is presented in section 4.7 in the following chapter. The new fluid property library called `TILFluids` is used in all models in the new component model library to analyze thermodynamic systems that is presented in chapter 4.

A simple valve model is used as a demonstrating example throughout this chapter to show the differences between the various approaches to model fluid properties. Code listing 3.1 shows the Modelica code for `PartialValve` that is used as base class for all further valve models. The equation to compute the density at the inlet is missing in this model and will have to be supplied in the derived models. Bernoulli's equation for compressible fluids is used to compute the mass flow rate in line 19 of code listing 3.1.

```
1  import SI = Modelica.SIunits;
2
3  connector SimpleFluidPort "Simple fluid port model"
4    SI.AbsolutePressure p "Pressure";
5    SI.SpecificEnthalpy h "Specific enthalpy";
6    flow SI.MassFlowRate m_flow "Mass flow rate";
7  end SimpleFluidPort;
8
9  partial model PartialValve "Partial base class for valve models"
10   SimpleFluidPort inlet "Inlet port";
11   SimpleFluidPort outlet "Outlet port";
12
13   SI.Density dInlet "Inlet density"; // an equation for dInlet has to be given
```

```
14                                    // in the derived classes
15 equation
16   inlet.m_flow + outlet.m_flow = 0 "Static mass balance";
17   inlet.h = outlet.h "Static energy balance";
18
19   inlet.m_flow = 0.2e-6*sqrt(2*dInlet*(inlet.p - outlet.p)) "Bernoulli's equation";
20 end PartialValve;
```

Code Listing 3.1: Modelica code common to all valve models used to demonstrate different approaches to compute fluid properties. An equation for the inlet density `dInlet` has to be given in the derived valve classes.

3.2 Function-Based Computation of Fluid Properties

The classical approach to compute fluid properties is purely function-based which means that unknown fluid properties are computed from known fluid properties using functions. The inlet density in the valve model presented in code listing 3.1 can be computed using a function that computes the density from the pressure and specific enthalpy at the inlet port. Code listing 3.2 shows the code for this approach. The function that is used to compute the density is taken from the function-based part of the `TILFluids` library that is explained in section 3.5. The third input to `density_ph()` specifies the name of the medium used for the computation.

```
import MediumFunctions = TILFluids.FunctionBasedMedium;

model ValveFB "Valve using function-based approach and TILFluids"
  extends PartialValve;
equation
  dInlet = MediumFunctions.density_ph(inlet.p, inlet.h, "IfTLibrary.R744");
end ValveFB;
```

Code Listing 3.2: Valve model with function-based approach to compute fluid properties using `TILFluids`.

The `Modelica.Media` library uses a slightly different function-based approach presented in code listing 3.3. The medium is specified by a replaceable local package called `Medium` using the concept of replaceable local classes as presented in section 2.4.2. The specific medium package used in code listing 3.3 is included in the `Modelica.Media`-compatible part of `TILFluids` (see section 3.5 for more information). It can be redeclared in the modifier of an instance of `ValveFBMedia`. The major difference to the approach presented in code listing 3.2 is the `stateInlet` that is used as input argument for the function to compute the density. The `stateInlet` is an instance of the `ThermodynamicState` record that is used in the `Modelica.Media` library to store the thermodynamic state of a medium. The record was initially designed to contain the minimal set of fluid properties required to compute all other fluid properties. For an ideal gas for example, the pressure and the temperature are the minimal set of fluid properties since all other properties can be computed from these two using the thermal equation of state (ideal gas law) and a correlation for the specific

heat capacity at constant volume (see Köhler, 2007a). The concept was slightly altered in some packages such as `WaterIF97_base` where more than the minimal set of fluid properties is stored in the `ThermodynamicState` record. The `stateInlet` is computed using the `setState_ph()` function defined in the medium package. The record is then used as input to the density function. The advantage of this concept is that the `ThermodynamicState` record has to be computed only once even if more properties than just the density are used in the component model. This is of great importance for the numerical efficiency since fluid property computations often require computationally expensive inverse iterations depending on the set of input variables and the variables used to compute the fluid properties (e.g., the set of variables stored in the `ThermodynamicState` record). Separate functions to compute the `ThermodynamicState` record are provided for the inputs density-temperature, pressure-specific enthalpy, pressure-specific entropy, and pressure-temperature. For mixtures, the composition vector X is required as a third input. The approach to compute fluid properties is still function-based despite the fact that instances of the `ThermodynamicState` record are created.

```
model ValveFBMedia "Valve using function−based approach and Modelica.Media"
  extends PartialValve;

  replaceable package Medium=TILFluids.Media.IfTLibrary.R744
   extends Modelica.Media.Interfaces.PartialMedium;
  Medium.ThermodynamicState stateInlet "Inlet state";
equation
  stateInlet = Medium.setState_ph(inlet.p, inlet.h);
  dInlet = Medium.density(stateInlet);
end ValveFBMedia;
```

Code Listing 3.3: Valve model with function-based approach to compute fluid properties using `Modelica.Media`.

The set of fluid properties in the `ThermodynamicState` record depends on the specific medium model. For an ideal gas, this set would be pressure and temperature as pointed out earlier. Many refrigerants are described using the Helmholtz equation of state (as suggested by McLinden et al., 1998, and others) usually given in its dimensionless form

$$\phi = \phi(\tau, \delta) \qquad \text{with} \quad \tau = \frac{T_c}{T} \quad \text{and} \quad \delta = \frac{\varrho}{\varrho_c} \tag{3.1}$$

where ϕ is the dimensionless Helmholtz energy, τ the dimensionless inverse temperature, and δ the dimensionless density. T_c and ϱ_c are the critical temperature and the critical density respectively. For refrigerants described by the Helmholtz equation, density and temperature are a good choice for the minimal set of fluid properties since all other fluid properties can be computed from this set of properties (see for example Span, 2000).

3.3 Object-Based Computation of Fluid Properties

Object-based approaches to compute fluid properties differ from function-based approaches in encapsulating all fluid property computations in a medium model. Each instance of those

medium models can be regarded as a state point in a thermodynamic plane for example in a pressure-enthalpy diagram. The medium models compute and store all thermodynamic and transport properties of interest.

The medium model is called `BaseProperties` in the `Modelica.Media` library. This model provides `3+nXi` equations for the `5+nXi` state variables as shown in table 3.1 where `nXi` is the number of independent mass fractions. Two variables out of `T`, `p`, `d`, `u`, and `h` and the mass fractions `Xi` are the independent variables. All other variables and the full mass fraction vector `X` are computed in the `BaseProperties` model. The medium model is defined in `PartialMedium` and is extended in the interface packages extending from this base class (see class diagram in figure 2.11).

	Variable	Unit	Description
State Variables	`T`	K	Temperature
	`p`	Pa	Absolute pressure
	`d`	kg/m^3	Density
	`u`	J/kg	Specific internal energy
	`h`	J/kg	Specific enthalpy
	`Xi[nXi]`	kg/kg	Independent mass fractions
	`ThermodynamicState`		Thermodynamic state record
	`X[nX]`	kg/kg	Mass fractions
	`R`	J/(kg K)	Gas constant
	`MM`	kg/mol	Molar mass
	`preferredMediumStates`	-	Boolean that is true if `StateSelect.prefer` shall be used for the independent property variables of the medium
	`T_degC`	°C	Temperature
	`p_bar`	bar	Absolute pressure
	`sat`		Saturation property record

Table 3.1: Variables in two-phase `BaseProperties` model.

Code listing 3.4 shows the code for a valve model using an instance of `BaseProperties` to compute all fluid properties. Like in code listing 3.3, the `Modelica.Media`-compatible part of `TILFluids` is used. The pressure and the specific enthalpy of the medium object `mediumInlet` are set in lines 8 and 9 in code listing 3.4. All other fluid properties are computed from these two properties and can be accessed directly as shown in line 11.

```
1 model ValveOBMedia "Valve using object−based approach and Modelica.Media"
2   extends PartialValve;
3
4   replaceable package Medium=TILFluids.Media.IfTLibrary.R744
5    extends Modelica.Media.Interfaces.PartialMedium;
6   Medium.BaseProperties mediumInlet "Medium object at inlet";
7 equation
8   mediumInlet.p=inlet.p;
```

```
9   mediumInlet.h=inlet.h;
10
11  dInlet = mediumInlet.d;
12 end ValveOBMedia;
```

Code Listing 3.4: Valve model with object-based approach to compute fluid properties using `Modelica.Media`.

From an object-oriented point of view it would be great to have several medium models computing different sets of variables. For example computing all transport properties in a medium object adds an unnecessary overhead if they are not used in the component model. Unfortunately, the `BaseProperties` model cannot be changed by the user to include additional variables since it is part of the Modelica Standard Library which is read-only. Instead the user can compute additionally required variables with functions defined in the interface class that take the `ThermodynamicState` record instantiated in `BaseProperties` as input. The specific entropy at the inlet of the valve which is not included in the `BaseProperties` model could be computed mixing the object-based approach with a function-based approach using the equation

```
s = Medium.specificEntropy(mediumInlet.state);
```

This mixing of different programming paradigms tends to be hard to understand for new users of the `Modelica.Media` library.

The `Modelica.Media` library uses replaceable local packages to make the medium model exchangeable as described in section 3.2. A much simpler approach would be to use the medium name as a string parameter in function calls as shown in code listing 3.2. The new object-based approach presented in section 3.5 also uses string parameters to specify the medium in each model.

3.4 Calling External Fluid Property Computation Codes

As already described in the introduction to this chapter, the `Modelica.Media` contains several ready-to-use medium models. However, there exists a large class of engineering systems for example refrigeration systems, heat-pump systems, or organic rankine cycles that require accurate models of application-specific two-phase fluids currently not included in the Modelica Standard Library.

A possibility to overcome this situation is to implement such medium models in Modelica possibly by conforming to the `Modelica.Media` interfaces for greater compatibility. The advantage of this approach is that self-contained Modelica models are obtained that can be optimized for efficiency by the simulator. The main drawback is that writing such code requires a sizable investment in terms of time and effort. The resulting code can then only be re-used in a Modelica context at least without any further significant effort. Additional medium packages extending the scope of the `Modelica.Media` library are for example implemented in the `ThermoFluidPro` library developed by the Swedish company Modelon and used as a base library in their `AirConditioning` library. Newer versions of the `ThermoFluidPro` library are

partly encrypted indicating another drawback of this approach which is the difficulty with protecting intellectual property rights when distributing the library.

Another possibility to overcome this situation is to take advantage of existing fluid property computation codes developed for general-purpose applications and to interface that code to Modelica. This approach becomes extremely attractive if the effort of developing those interfaces to external fluid property computation codes is kept to a bare minimum. This was the major objective for the new `Modelica_ExternalMedia` library developed in close cooperation with Prof. Casella from Politecnico di Milano. Currently only two-phase single-substance fluids have been considered since this combination already covers many interesting applications for Modelica that cannot be developed using existing `Modelica.Media` models. The goals of the `Modelica_ExternalMedia` library can be summarized as follows:

- 100% compatibility with the `Modelica.Media` interface
- compatibility with multiple Modelica tools and C/C++ compilers
- small effort required to develop the interface to additional external fluid property computation codes
- computational efficiency compatible with Modelica-internal solutions

The entire project consists of three parts: the Modelica library `Modelica_ExternalMedia`, an interface layer written in C, and an object-oriented interface to external fluid property computation codes written in C++ called ExternalMedia. The project including the Modelica and the C/C++ source code is released under the Modelica license[1]. The current implementation was compiled and tested using different C++ compilers such as Microsoft Visual C++ and MinGW. Figure 3.1 shows a class diagram of parts of `Modelica.Media` and the `Modelica_ExternalMedia` library. A set of functions in the `ExternalTwoPhaseMedium` package corresponds one-to-one to the C interface layer functions. These functions are called according to the external function mechanism as defined in the Modelica specification (Modelica Association, 2005). The interface layer functions in turn manage a collection of C++ objects that define the interface to the external fluid property computation codes.

A very important aspect when developing an interface to external fluid property computation codes is the communication between the Modelica models and the external library. `Modelica_ExternalMedia` and the ExternalMedia use a unique identification number `uniqueID` to assign an instance in Modelica to a medium instance in the C/C++ layer. This identification number is zero by default and can be positive or negative depending on the method to compute fluid properties. The underlying mechanisms are explained in more detail by Richter and Casella (2008).

3.4.1 Architecture of the Modelica Layer (`Modelica_ExternalMedia`)

The class diagram shown in figure 3.1 can be divided into three different layers. The first layer contains the interface classes in `Modelica.Media`. The second layer contains the in-

[1] The Modelica License can be found at http://www.modelica.org/libraries/Modelica/ModelicaLicense.html.

terface class in `Modelica_ExternalMedia` that extends from `PartialTwoPhaseMedium`. The third layer contains the ready-to-use medium models in `Modelica_ExternalMedia`. The `ExternalTwoPhaseMedium` package is a generic package and defines a string constant that is used to specify the actual external code to be used in the derived packages. By default a `TestMedium` implemented in C++ is provided that roughly assembles the properties of cold water at low pressure and that can be used as a starting point when implementing interfaces to external fluid property computation code using the C/C++ layer of the library.

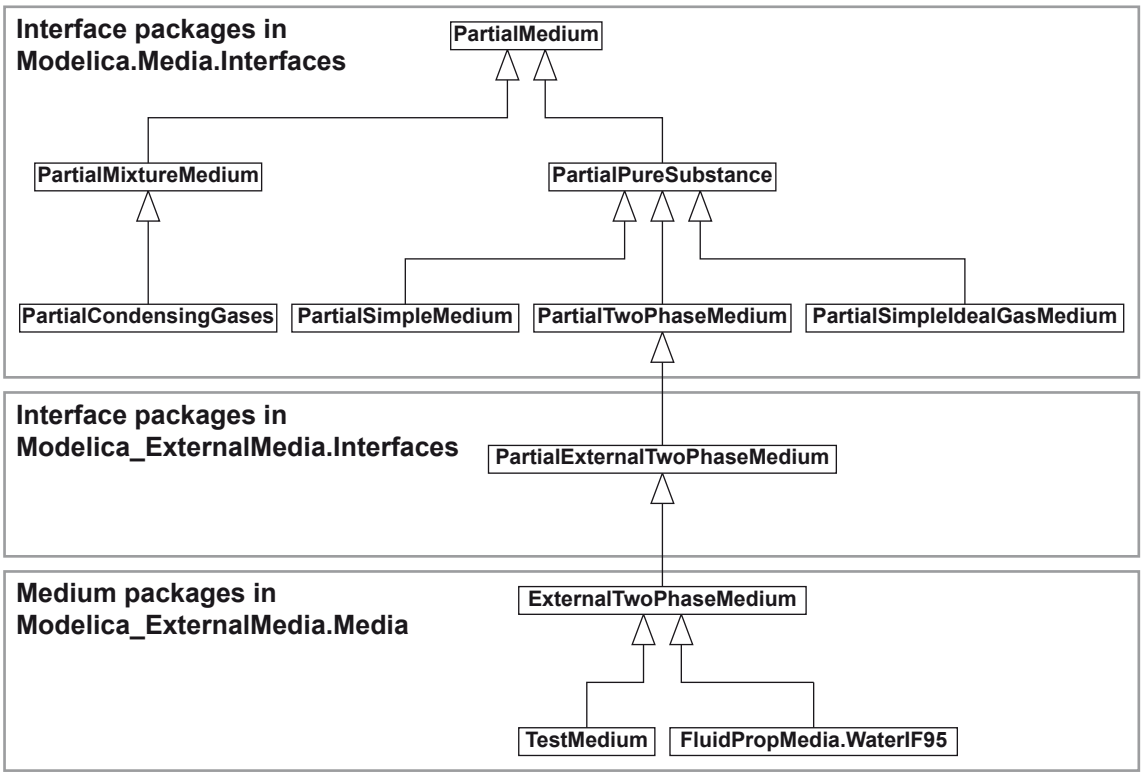


Figure 3.1: UML class diagram of medium interface classes of `Modelica.Media` and `Modelica_ExternalMedia`. The `TestMedium` package is a demo implementation roughly assembling the properties of cold water at low pressure that can be used as a starting point when developing new interfaces to external fluid property computation codes.

The interface package `PartialExternalTwoPhaseMedium` in the second layer in figure 3.1 is required for the following three reasons:

1. Redeclaration of Functions with Default Implementations

`PartialExternalTwoPhaseMedium` redeclares all functions from `PartialTwoPhaseMedium` that are provided with a default implementation. The functions have to be redeclared so that they do not contain an algorithm and an external algorithm section which would not be legal Modelica. The function interfaces are extended to include the `uniqueID` as an input which is zero by default.

2. Extension of the `ThermodynamicState` and `SaturationProperties` Records

The only component of the `ThermodynamicState` record of a `PartialTwoPhaseMedium` is an integer specifying the phase. The phase integer can be an input to the `setState_XXX()` functions as well as an output depending on its value. Setting phase to zero means that the user does not know the phase in advance. The medium model is then determining the phase and returns the value one for a one-phase state and two for a two-phase state. Setting phase to one when calling a `setState_XXX()` function forces the state of the medium to be one-phase while setting it to two forces the state to be two-phase. `PartialExternalTwoPhaseMedium` adds the `uniqueID` to the `ThermodynamicState` record along with the pressure, temperature, density, specific entropy, and specific enthalpy. The `uniqueID` is important to identify the medium instance in the C/C++ interface layer as explained in section 3.4.2.

The `SaturationProperties` record is defined in `PartialTwoPhaseMedium` and contains the saturation pressure and the saturation temperature. The saturation record is extended in `PartialExternalTwoPhaseMedium` to include the `uniqueID` that is again used to identify the correct instance when communicating with the C/C++ interface library.

Including the unique identification number in the `ThermodynamicState` record as well as in the `SaturationProperties` record ensures that additionally computed fluid properties not contained in `BaseProperties` are returned from the correct medium instance in the C/C++ interface library and that those two records also work when using the function-based approach explained in section 3.2.

3. Extension of the `BaseProperties` Model

The `BaseProperties` model in `PartialTwoPhaseMedium` contains the variables listed in table 3.1. The model is extended in `PartialExternalTwoPhaseMedium` to include the additional variables shown in table 3.2.

Variable	Unit	Description
`basePropertiesInputChoice`	-	Enumeration to specify the input variables for the property computation (e.g., ph or pT)
`phaseInput`	-	Phase input for property computation functions (2 for two-phase, 1 for one-phase, 0 if unknown)
`phaseOutput`	-	Phase output for medium (2 for two-phase, 1 for one-phase, 0 if not known)
`uniqueID`	-	Unique ID number
`s`	J/kgK	Specific entropy

Table 3.2: Additional variables in external two-phase `BaseProperties` model.

The `BaseProperties` model creates a medium instance in the C++ layer when it is initialized using the `createMedium()` function. The return value of this function is the unique identification number of the `BaseProperties` instance that is stored in `uniqueID`. This num-

ber is used in all further function calls that compute fluid properties in the `BaseProperties` model.

3.4.2 Architecture of the C/C++ Layer (ExternalMedia)

The C/C++ layer ExternalMedia manages a collection of objects that define the interfaces to the external fluid property computation codes as shown in figure 3.2. The first fundamental object is the Solver object which encapsulates the external fluid property computation code. In order to manage several different solvers at the same time a *SolverMap* is defined. The Solver objects in this map are indexed using the strings defined in the `ExternalTwoPhaseMedium` package. Each time an external function is called, the strings for the medium name, the library name, and the substance names are passed as parameters so that the corresponding solver can be instantiated when the function is called for the first time. This allows the interface layer to always point to the correct solver.

The second fundamental object is the Medium object which corresponds to a point on a thermodynamic plane or on the saturation curve for saturation properties. Each Medium object contains a pointer to the corresponding Solver object and a record of type *TwoPhaseMediumProperties* which contains the values of all the thermodynamic properties and is used as a cache record. All instances of Medium objects are stored in the *MediumMap* which is indexed using the *uniqueID*.

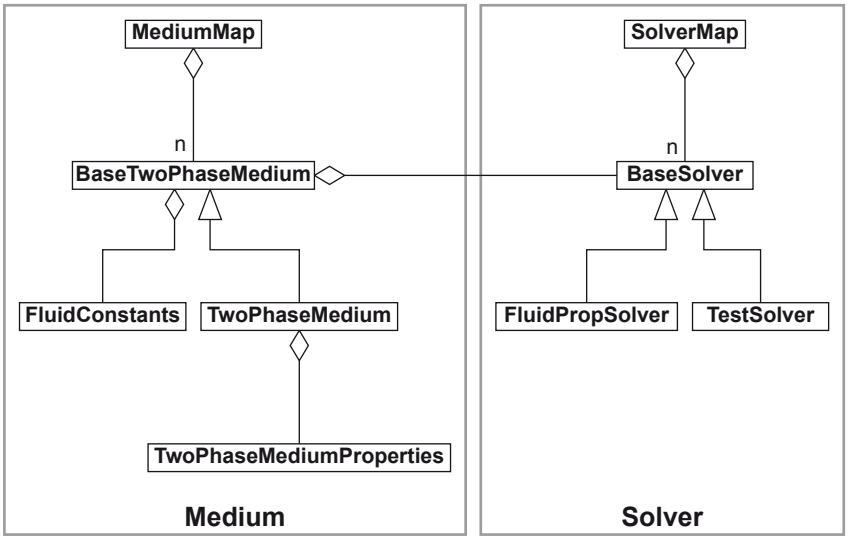


Figure 3.2: UML class diagram of the ExternalMedia library which is the C/C++ layer to the `Modelica_ExternalMedia` library.

3.5 Advanced Object-Based Computation of Fluid Properties

The previous section explained how the fluid property library `Modelica.Media` included in the Modelica Standard Library was extended to support external fluid property computation codes in Modelica. This extension widens the scope of the fluid property library in terms of the number of engineering problems that can be solved using it. Nevertheless, there are some

additional drawbacks of the library that cannot be overcome without major changes to its design.

The first major drawback already discussed in section 2.7 is demonstrated in the two class diagrams shown in figure 2.11 and 3.1: The object-oriented structure of the `Modelica.Media` library and the `Modelica_ExternalMedia` library is rather complex. Some basic functionality is defined in `PartialMedium` and additional functionality is added throughout the entire inheritance tree making it hard to find the position of any specific implementation. Some people might argue that this is a tool issue and that the current Modelica tools are just not good enough in browsing through inheritance trees. While this argument is true, it is also always desirable to create a library that can easily be understood without having to use advanced tools.

Another major drawback of the `Modelica.Media` library is that it tries to fit many medium models in one single framework: incompressible liquids, real gases, two-phase fluids, and mixtures. On the one hand, this is a very unique concept allowing users to switch from a simple water model such as `ConstantPropertyLiquidWater` to a state-of-the-art model like `WaterIF97_pT` just by redeclaring the medium package in their models. On the other hand, this approach further complicates writing component models especially for user not familiar with the structure of the `Modelica.Media` library.

In addition to these two drawbacks, the object-based approach in `Modelica.Media` using the `BaseProperties` model is not flexible enough as pointed out in section 3.3 regarding the development of user-specific medium models. The medium models implemented in Modelica can furthermore only be used in a Modelica context at least without any further significant effort. This drawback is partially overcome by the `Modelica_ExternalMedia` library that can also only be used in Modelica.

The above mentioned drawbacks were the starting point for the development of a new object-based model library for the computation of fluid properties. The resulting Modelica library `TILFluids` is used in all component models presented in the following chapter. It can be split into three parts: a function-based library, a 100% `Modelica.Media`-compatible library, and an object-based library consisting of a set of medium objects for gases, liquids, refrigerant, and moist air. The library is described in section 3.5.1.

`TILFluids` uses the external library TILFluidsLib to handle solver and medium objects. This external library that is written in C++ is explained in section 3.5.2. Section 3.5.3 presents the two most important external fluid property computation codes interfaced from the TILFluidsLib.

3.5.1 Architecture of the Modelica Layer (`TILFluids`)

`TILFluids` offers three different approaches to compute fluid properties in Modelica. The first approach is purely function-based and can be found in `FunctionBasedMedium` and was used in code listing 3.2. The package `FunctionBasedMedium` contains a set of equations to compute all important thermodynamic and transport properties from dT, ph, ps, and pT requiring the medium name as a third input. A possible application of the function-based

approach within an object-oriented component model library is the computation of fluid properties during initialization before any medium object is created.

The second approach can be found in `Interfaces` and `Media` and offers medium packages fully `Modelica.Media`-compatible. Figure 3.3 shows the class structure for this part of the library. The `PartialTILMedium` extends from `PartialExternalTwoPhaseMedium` and redefines the linked library from ExternalMedia to TILFluidsLib. This ensures that `TILFluids` can be used together with other media libraries based on the ExternalMedia library. All other medium packages extend from `PartialTILMedium` and can be used as a replacement for medium models based on `PartialTwoPhaseMedium`. Offering a `Modelica.Media`-compatible interface allows users to use `TILFluids` in their `Modelica.Media`-based projects thus widening the scope of the new fluid property library.

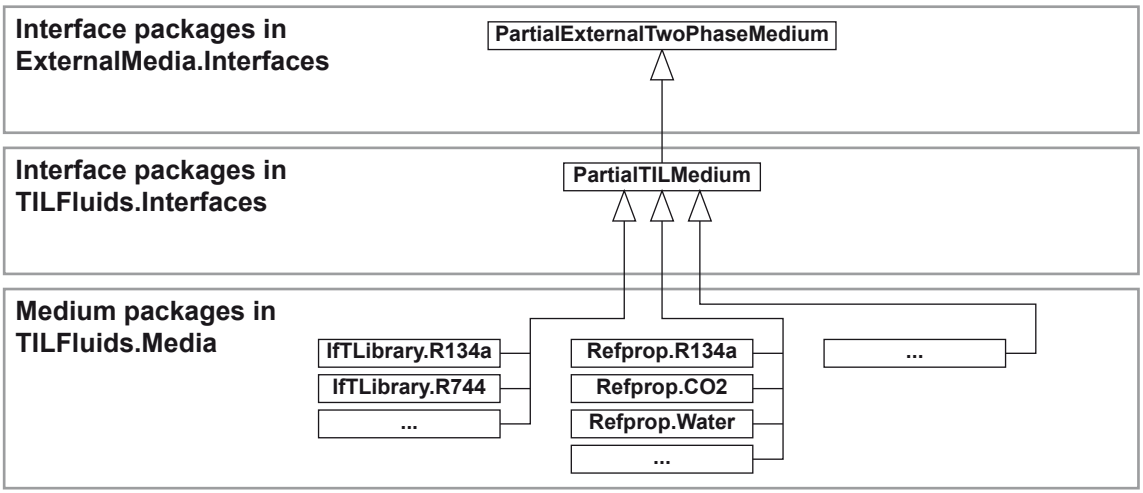


Figure 3.3: UML class diagram of `Modelica.Media`-compatible part of `TILFluids`.

The third and most important approach is the new object-based approach consisting of a couple of medium models in the top level of `TILFluids`. Table 3.3 briefly describes the available medium models for gases, liquids, refrigerant, and moist air. The medium models in `TILFluids` are very flexible and can be tailored to the specific requirements of a component model. The medium objects in a simulation can also be used to automatically generate thermodynamic plots of the simulation results as described in section 5.3.

Code listing 3.5 shows the valve model implemented using a `PortRefrigerant` object from `TILFluids`. The medium to be used is specified in the modifier by settings the refrigerant name to `IfTLibrary.R744`. This seems to be much simpler to understand than the replaceable local package that is used with `Modelica.Media` to redeclare the medium (see code listing 3.4 for example).

```
model Valve "Valve using object−based approach and TILFluids"
  extends PartialValve;

  TILFluids.PortRefrigerant inletRefrigerant(refrigerantName="IfTLibrary.R744")
   "Medium object at inlet";
equation
```

```
  inletRefrigerant.p = inlet.p;
  inletRefrigerant.h = inlet.h;

  dInlet = inletRefrigerant.d;
end Valve;
```

Code Listing 3.5: Valve model with object-based approach to compute fluid properties using `TILFluids`.

All medium models available in `TILFluids` are listed in table 3.3. The models contain different sets of variables depending on the type of the medium as well as on the model itself. This enables the developer to use medium models depending on the required fluid properties to improve the numerical efficiency of component models (e.g., by using a medium that does not compute the transport properties if those are not required in the component model).

Name	Description
`Gas`	Base model for all gases. This model computes density, specific enthalpy, pressure, temperature, and additional properties from dT, ph, or pT. It uses the *TILFluidsGasSolver* for all computations.
`PortGas`	Gas model specifically designed to be used at ports of components. This model computes all properties from ph and uses the *TILFluidsGasSolver*.
`Liquid`	Base model for all liquids. This model computes density, specific enthalpy, temperature, specific heat capacity, and additional properties from T or h. It uses the *TILFluidsLiquidSolver* for all computations.
`PortLiquid`	Liquid model specifically designed to be used at ports of components. This model computes all properties from h and uses the *TILFluidsLiquidSolver*.
`Refrigerant`	Base model for all refrigerants. This model computes density, specific enthalpy, pressure, specific entropy, temperature, and additional properties from dT, ph, ps, or pT. It uses the *TILFluidsSolver* for all computations.
`SimpleRefrigerant`	This is a lightweight model that can be used as a replacement for `Refrigerant` if less additional properties are required.
`PortRefrigerant`	Refrigerant model specifically designed to be used at ports of components. This model computes all properties from ph and uses the *TILFluidsSolver*.
`MoistAir`	Moist air model that computes all moist air properties from $ph_{1+x}x_w$, $ps_{1+x}x_w$, pTx_w, $pT\phi$, $pT\xi$, and $ph\xi$ where h_{1+x} and s_{1+x} are the specific enthalpy and specific entropy per kilogram of dry air respectively, x_w is the water content, ϕ is the relative humidity, and ξ is the steam concentration.

Table 3.3: Medium models in `TILFluids`.

Table 3.4 shows the variables contained in each refrigerant model. The `Refrigerant` model is the most advanced model containing the full range of fluid properties including saturation and transport properties. The `SimpleRefrigerant` is a lightweight model that does not contain saturation properties, transport properties, and critical properties. Both models support the full range of input variables: dT, ph, ps, and pT. The `PortRefrigerant` model is specifically designed to be used to compute fluid properties at connectors. It contains almost the same variables as the `SimpleRefrigerant` model.

Variable	Unit	Description	R	PR	SR
d	kg/m^3	Density	X	X	X
h	J/kg	Specific enthalpy	X	X	X
p	Pa	Pressure	X	X	X
s	J/(kg K)	Specific entropy	X	X	X
T	K	Temperature	X	X	X
u	J/kg	Specific internal energy	X		
x	kg/kg	Steam mass fraction	X	X	X
cp	J/(kg K)	Specific heat capacity at constant pressure	X	X	X
cv	J/(kg K)	Specific heat capacity at constant volume	X		
T_degC	°C	Temperature	X	X	X
beta	1/K	Isothermal expansion coefficient	X	X	X
kappa	1/Pa	Compressibility	X	X	X
drhodh	m^3/J	Derivative of density wrt specific enthalpy	X		
drhodp	m^3/(kg Pa)	Derivative of density wrt pressure	X		
sat		Saturation property record	X		
Tsat	K	Saturation temperature			X
transp		Transport property record	X		
mm	kg/mol	Molar mass	X		
Ri	J/(kg K)	Specific gas constant	X		
crit		Critical data record	X		

Table 3.4: Variables in refrigerant models in `TILFluids` (`R` = `Refrigerant`, `PR` = `PortRefrigerant`, and `SR` = `SimpleRefrigerant`).

3.5.2 Architecture of the C/C++ Layer (TILFluidsLib)

The C/C++ layer that is used by `TILFluids` is called TILFluidsLib. The TILFluidsLib uses the source code of the ExternalMedia library presented in section 3.4.2 and extends it by inheritance and by adding classes. The most important changes are the extension of the cache record in *TILFluidsMediumProperties*, the improvement of the solver in *TILFluidsSolver* that allows determining the valid region for each medium model, and the introduction of new medium and solver classes to handle various medium types such as gases or liquids. The extension to other medium types is only added as an afterthought in the current implementation and could be improved in a future version. TILFluidsLib also introduces a

powerful logging mechanism that allows tracing errors and detecting possible inefficiencies when working with the interface library.

The most important change compared to the ExternalMedia library is that the TILFluidsLib does not only provide the Modelica interface `TILFluids` but also interfaces to other software applications. The most important interfaces allow using the TILFluidsLib in MS Excel, MATLAB, Simulink, and many other software tools utilizing one of the more generic interfaces provided in C, C++, and Python. The TILFluidsLib is also used to generate the phase diagrams for the automated thermodynamic visualization described in section 5.3. The additional interfaces allow to use external fluid property computation codes in a number of software tools by implementing a single interface in the TILFluidsLib. This is a major advantage compared to a Modelica-internal solution that cannot easily be accessed from other software tools.

3.5.3 Interfaced External Fluid Property Computation Codes

This section describes the two most important external fluid property computation codes that are included in the TILFluidsLib: The IfTLibrary and REFPROP. The IfTLibrary is an extraction of the fluid property code from the simulation platform for thermodynamic systems developed by Tegethoff (1999). The code is written in C++ and compiled directly into `TILFluids` making it independent from the operating system. Table 3.5 lists all fluids available in the IfTLibrary and gives some additional information on the implemented set of equations.

Refrigerant	R134a	Helmholtz equation	Coefficients taken from Tillner-Roth et al. (1998), transport properties based on Krauss et al. (1993)
	R744	Helmholtz equation	Coefficients taken from Span and Wagner (1996), transport properties based on Vesovic et al. (1990)
Gas	DryAir	Ideal Gas Law	Transport properties fitted to data from VDI (2002)
Liquid	H2O		Properties fitted to data from VDI (2002)
	H2O/Propylenglykol		Properties fitted to data from VDI (2002)

Table 3.5: Fluids available in the IfTLibrary.

Many simulation tools for refrigeration, air-conditioning, and heat-pump systems use or at least supply an interface to the REFPROP fluid property database. REFPROP stands for Reference Fluid Thermodynamic and Transport Properties Database and is developed and maintained at the National Institute of Standards and Technology (NIST) by Lemmon et al. (2007). REFPROP features a graphical user interface that supports the computation of fluid properties and that provides simple thermodynamic plots such as pressure-enthalpy and temperature-entropy diagrams.

Medium		Equation of State
R134a	FEQ	Helmholtz equation of state for R134a of Tillner-Roth & Baehr (1994)
	FES	Short Helmholtz equation of state for R134a of Span and Wagner (2003)
	BWR	MBWR equation of state for R134a of Huber and McLinden (1992)
	FE2	Helmholtz equation of state for R134a of Astina and Sato (2004)
	PRT	Translated Peng-Robinson equation
CO_2	FEQ	Helmholtz equation of state for carbon dioxide of Span and Wagner (1996)
	FEK	Helmholtz equation of state for carbon dioxide of Kunz and Wagner (2004)
	BWR	MBWR equation of state for carbon dioxide of Ely et al. (1987)
	FE1	Helmholtz equation of state for carbon dioxide of Ely et al. (1987)
	FES	Short Helmholtz equation of state for carbon dioxide of Span and Wagner (2003)

Table 3.6: Equations of state for R134a and CO_2 in REFPROP.

Four different equations of state are implemented in REFPROP to compute the thermodynamic properties of fluids: the modified Benedict-Webb-Rubin equation of state (MBWR), the Helmholtz form of the pure fluid equation of state, a volume translated Peng-Robinson equation, and an extended corresponding states model (ECS model) with temperature and density dependent shape factors. REFPROP allows the user to choose the used equation of state depending on the selected medium. This feature offers the possibility to compare different equations of state with regard to accuracy vs. computing time. Table 3.6 lists all equations of state available in REFPROP for R134a and CO_2.

`TILFluids` can take full advantage of the different models implemented in REFPROP by specifying additional options in the medium string. All specific models from REFPROP that can be specified in the medium string are listed in table 3.7. The medium string `"Refprop.CO2(EOS=FES,ETA=VS4).fld"` for example specifies a medium CO_2 from REFPROP using the short Helmholtz equation of state of Span and Wagner and the pure fluid generalized friction theory viscosity model of Quiones-Cisneros and Deiters.

	Model		Model
EOS	Equation of state	STN	Surface Tension
ETA	Viscosity	MLT	Melting Line
TCX	Thermal Conductivity	SBL	Sublimation Line

Table 3.7: Models in REFPROP that can be selected in the medium string from `TILFluids`.

3.5.4 Comparison of Computational Efficiency

The computation of fluid properties accounts for a large part of the total required computation time when simulating thermodynamic systems. The computational efficiency of fluid

property libraries is therefore of great interest. A simple test was set up to demonstrate the time requirements for different fluid property implementations. The medium model used for the comparison only contains density, specific enthalpy, pressure, specific entropy, and temperature. CO_2 was selected as medium. The specific enthalpy was kept constant at 300 kJ/kg and the pressure was varied from 20 to 140 bar according to the first diagram in figure 3.4. The other three diagrams show the density, specific entropy, and temperature respectively.

Three different medium libraries were used for the comparison: the IfTLibrary (see table 3.5), the REFPROP library with the default equation of state (FEQ) and with a simpler equation of state (FES) (see table 3.6), and the `ThermoFluidPro` library developed and maintained by Modelon which also uses a Helmholtz equation. The test was performed using a fixed step Euler solver to ensure an identical number of function calls without any influence by a step-size control. 3,000 steps were used for all examples. The medium models in `TILFluids` were simulated with and without the caching functionality in the TILFluidsLib. For each evaluation, all fluid properties including the transport properties were externally computed for the media from `TILFluids`.

From the execution times listed in table 3.8, it can be seen that the external fluid codes are not significantly slower than the implementation in Modelica. It can also be seen that the cache record works very efficiently. The cache record stores all fluid properties computed after each inverse iteration. This means that an inverse iteration only has to be performed once at each time step and not three times (for computing the density, the specific entropy, and the temperature) as required without the cache. No inverse iteration is needed from 100 s to 200 s since the inputs pressure and specific enthalpy are constant yielding execution times that are about 22% of the execution times without using the cache record.

Library	**Medium Name**	**Time**
`TILFluids`	`"IfTLibrary.R744"`	1.31 s
	`"Refprop.CO2.fld"`	2.58 s
	`"Refprop.CO2(EOS=FES).fld"`	0.515 s
`TILFluids`	`"IfTLibrary.R744"`	5.52 s
without caching in external library	`"Refprop.CO2.fld"`	16.2 s
	`"Refprop.CO2(EOS=FES).fld"`	2.3 s
`ThermoFluidPro`	`Media.Technical.Co2`	1.38 s

Table 3.8: Comparison of computational efficiency of different implementations to compute fluid properties.

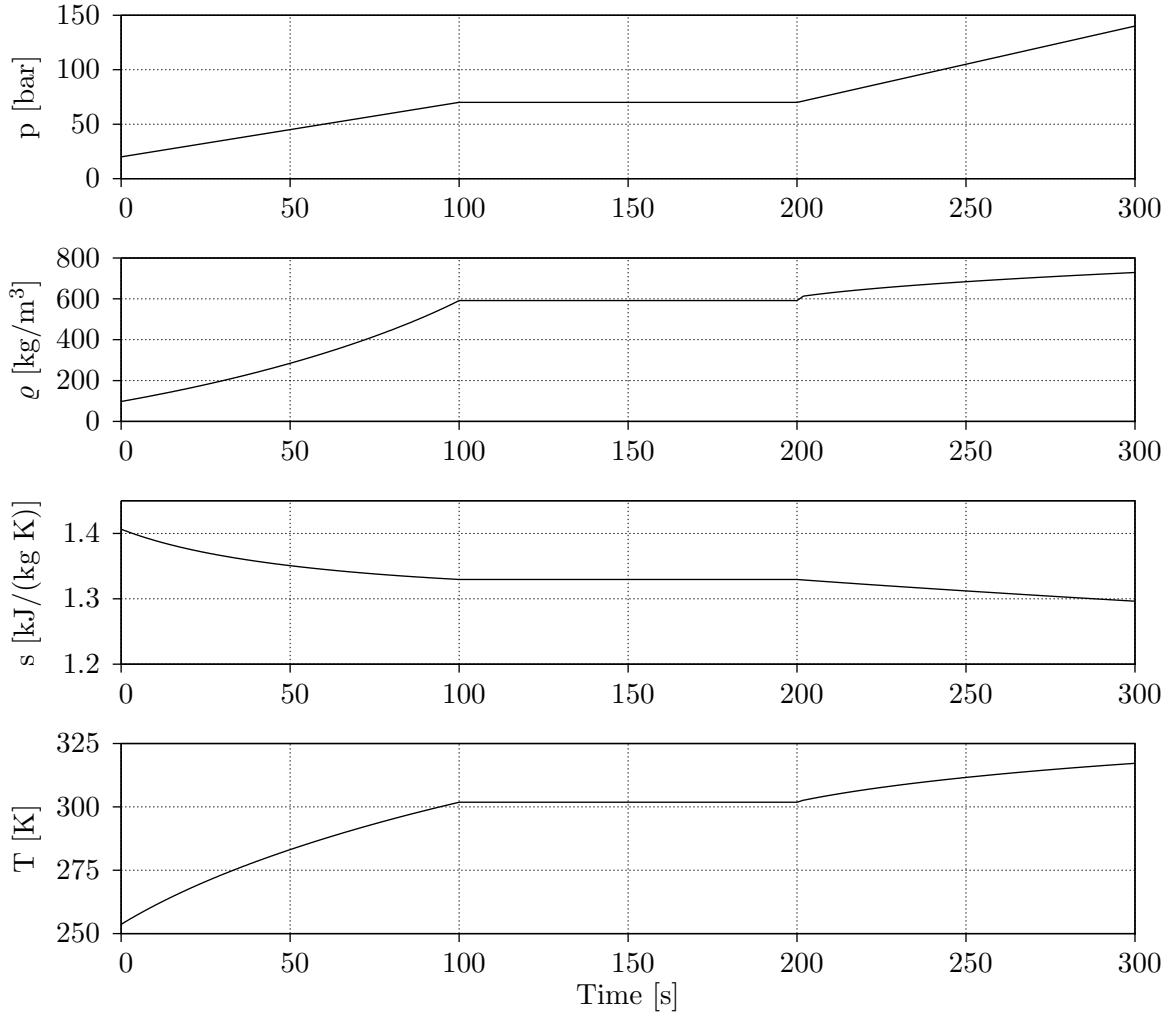


Figure 3.4: Medium properties of CO_2 over time for comparison of the computational efficiency of different implementations. The pressure was varied according to the topmost diagram while the specific enthalpy was kept constant at $h = 300$ kJ/kg. The test was performed using a fixed step Euler solver with 3,000 steps.

Chapter 4

Object-Oriented Modeling of Fluid Systems

This chapter describes the new object-oriented equation-based component model library for the simulation of refrigeration, air-condition, and heat-pump systems. The structure of the new library is simple to understand and allows for various extensions to cover a wide range of thermodynamic applications. This chapter describes the basic component models as well as the formulation of the balance equations. It also explains the structure of more complex component models such as the heat exchanger model.

4.1 Introduction

This chapter presents the new object-oriented component model library developed within the scope of this thesis. It provides component models to simulate refrigeration, air-conditioning, and heat-pump systems and can easily be extended with user-defined models to cover a wide range of thermodynamic applications. The main focus of this chapter is to present the basic design principles of the library such as the hierarchical structure and the basic control volumes. Many component models are available in the new library or in one of the project-specific add-on libraries. Some of these models were implemented within the scope of this work whereas other models were implemented by a team of developers demonstrating the sustainability of the chosen approach. The structure of the new component model library is easy to understand and follows the design rules presented in chapter 2. The flexible structure is one of the key features of the new component model library and distinguishes it from other object-oriented Modelica libraries in the field of thermodynamic system simulations. The new component model library is called `TIL` which is short for TLK-IfT-Library. The name reflects the significant contribution of both partners, the Institut für Thermodynamik and the TLK-Thermo GmbH, that both work with the new component model library and extend it with additional component models.

Figure 4.1 shows a CO_2 air-conditioning system composed of models from the new component model library. The shown system is used as a demonstrating example throughout this

chapter. The refrigeration system features a compressor, two gas coolers in series connected by a tube, a valve, an evaporator, and a low-pressure accumulator. In addition to these components representing real-world objects, additional components are used in the refrigeration system. A system information manager called `sim` specifies all important system information such as the used medium models and collects information like the total refrigerant mass in the system. Two pressure state components are used to compute the time derivative of pressure at the high and at the low pressure side respectively. These components are required for the simplification of the conservation laws explained in section 4.3. The `StateViewerInterface` in front of the compressor allows for an automated generation of thermodynamic phase diagrams as explained in section 5.3. Sink and source components are used to represent the air-side boundary conditions for the three heat exchangers.

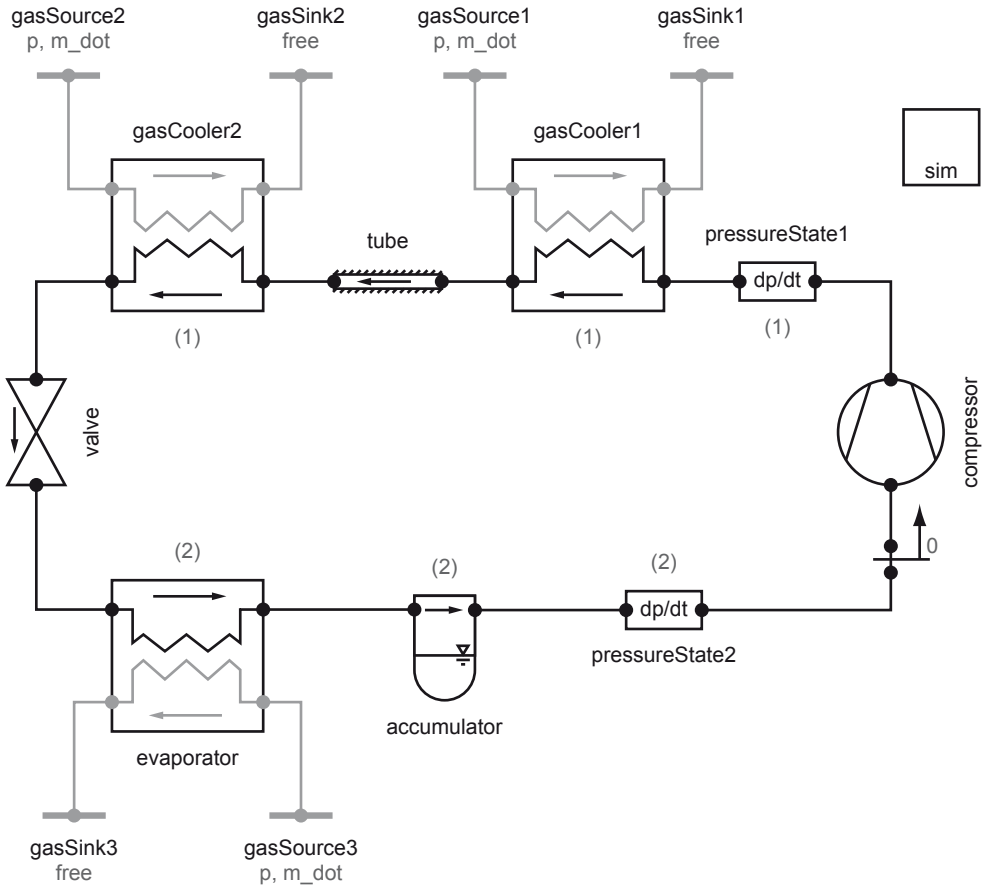


Figure 4.1: CO_2 refrigeration system using component models from the new component model library `TIL`.

This chapter presents all component models used in the air-conditioning system that is shown in figure 4.1. Section 4.2 explains the connectors and the connection equations that are of great importance for all component models. The conservation laws are derived in section 4.3 in a general form. Specific implementations of the conservation laws are presented in section 4.4 for the accumulator model and in section 4.7 for the basic control volumes used to assemble more complex models such as the heat exchanger models. Section 4.5

describes the heat transfer and pressure drop models used in the basic control volumes. The heat exchanger model itself is explained in section 4.8. Section 4.10 describes some additional component models. An advanced model for a swash plate compressor can be found in section 4.9. Some numerical aspects concerning the CO_2 refrigeration system shown in figure 4.1 are presented in section 4.11. Further applications of the new component model library are described in chapters 6 and 7.

Figure 4.2 shows the structure of the new component model library. The `HeatTransfer` package contains base models for heat transfer correlations for fluids and for solids and the `PressureDrop` package contains model for pressure drop correlations for fluids. The provided models are further specialized on the component model level. The `Material` package contains models for solids such as Aluminium, Steel, or Copper. Some general utility functions are provided in the `Utilities` package. The `HVAC` (Heating, Ventilation, and Air-Conditioning) package contains simple steady-state component models. This part of the new component model library is explained in more detail in appendix C. It was successfully used as a design tool for an ejector test bench as presented in Richter et al. (2006) and is also used for educational purposes in university lectures and commercial training courses. The example in section 5.4 also uses a simple system from this part of the new component model library. The by far most important part of the new library is the `HVAC_p` package that contains component models for the transient simulation of refrigeration, air-conditioning, and heat-pump systems. The `_p` in the name of the package indicates the specific simplification of the balance equations that is explained in section 4.3. Figure 4.2 also shows the structure of the `HVAC_p` package.

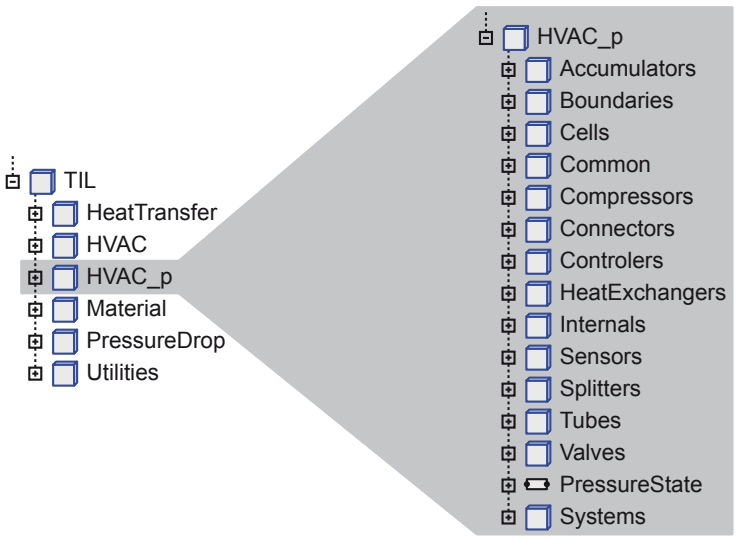


Figure 4.2: Structure of the new component model library `TIL` and its most important package `HVAC_p`.

4.2 Connectors and Connection Equations

A very important feature of Modelica are the connectors which are instances of connector classes. The connector classes define the variables that are part of the communication inter-

face that is specified by a connector. Understanding the connector design and the resulting connection equations is of great importance to avoid some common mistakes when dealing with fluid systems that are described in this section. The fluid connector in the new component model library is designed following the fluid connector design proposed by the Modelica Association in the `Modelica_Fluid` library (Elmqvist et al., 2003). Code listing 4.1 shows the code for the fluid connector called `FluidPort` in the new component model library.

```
connector FluidPort "Fluid port"
  parameter Integer nc=1 "Number of components";
  parameter String mediumName "Medium name";
  Integer index "Index for StateViewer";

  SI.AbsolutePressure p "Pressure";
  flow SI.MassFlowRate m_flow "Mass flow rate";

  SI.SpecificEnthalpy h "Specific enthalpy";
  flow SI.EnthalpyFlowRate H_flow "Enthalpy flow rate";

  SI.MassFraction xi[nc-1] "Independent mixture mass fractions m_i/m in the connection
    point";
  flow SI.MassFlowRate mXi_flow[nc-1] "Mass flow rates of the independent substances
    from the connection point into the component (mXi_flow = m_flow*Xi if m_flow > 0)";
end FluidPort;
```

Code Listing 4.1: Modelica code for `FluidPort` from `Connectors` package.

A connector can contain two different kinds of variables: non-flow variables representing a potential or an energy level and flow variables representing some kind of a flow. Equality coupling is applied for the non-flow variables and sum-to-zero coupling for the flow variables. A flow is defined to be positive when it flows into the component. One of the difficulties when simulating fluid systems is the handling of flow reversal or zero-flow situations. The enthalpy flow rate $\dot{H}$ in the connector can be computed as follows

$$\dot{H} = \begin{cases} \dot{m}h_1 & \dot{m} > 0 \\ \dot{m}h_2 & \text{otherwise} \end{cases} \tag{4.1}$$

assuming that h_1 is the specific enthalpy for positive mass flow rates and h_2 the specific enthalpy in the case of negative mass flow rates. Figure 4.3 shows the corresponding diagrams for the enthalpy flow rate $\dot{H}$ and the specific enthalpy h respectively.

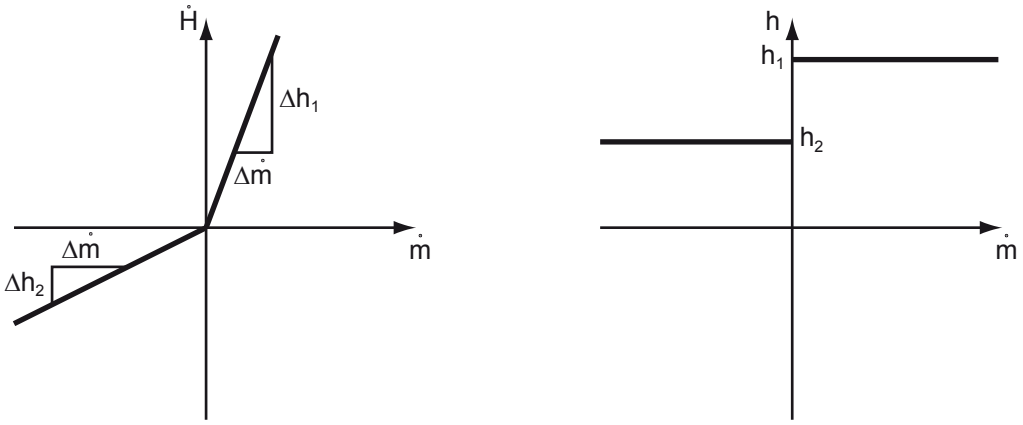


Figure 4.3: Enthalpy flow rate and specific enthalpy during flow reversal according to equation (4.1).

Note the discontinuity of the specific enthalpy h at zero mass flow rate in figure 4.3. The solver has to keep the previous value of the specific enthalpy h in case of zero flow that depends on the previous flow direction. In Modelica, the function `semiLinear()` was introduced to support the solver in detecting flow reversals. In fluid systems, `semiLinear()` is used to compute the enthalpy flow rate over the component boundary according to

```
port.H_flow = semiLinear(port.m_flow, port.h, h);
```

where `port` is an instance of type `FluidPort` (see code listing 4.1) and `h` is the specific enthalpy within the component. The `semiLinear()` function can be written as shown in equation (4.1) yielding an if-then-else expression in Modelica. Using `semiLinear()` instead allows tools to perform advanced checks and symbolic simplifications on connection equations. The resulting function for the enthalpy flow rate is plotted in figure 4.3 where h_1 refers to `port.h` and h_2 to `h`.

An interesting special case is the connection of three (or more) connectors using the `semiLinear()` function. Figure 4.4 shows the connection of three components to each other. For reasons of simplicity, only the pressure p, the specific enthalpy h, the mass flow rate $\dot{m}$, and the enthalpy flow rate $\dot{H}$ are considered. The independent mixture mass fractions ξ and their mass flow rates $\dot{m}_\xi$ could be treated in the same way.

The connection itself can be treated like an infinitesimal control volume at the connection point. The resulting equations are (taking into account the assumed flow directions)

$$p = p_W = p_E = p_N \tag{4.2a}$$

$$\dot{m}_W + \dot{m}_E = \dot{m}_N \tag{4.2b}$$

$$h = h_W = h_E = h_N \tag{4.2c}$$

$$\dot{H}_W + \dot{H}_E = \dot{H}_N \tag{4.2d}$$

In each of the three components W, E, and N the `semiLinear()` function is used to compute the enthalpy flow rate across the component boundary. The problem arises when looking at the values of the specific enthalpy in the connectors itself which is always the mixing enthalpy

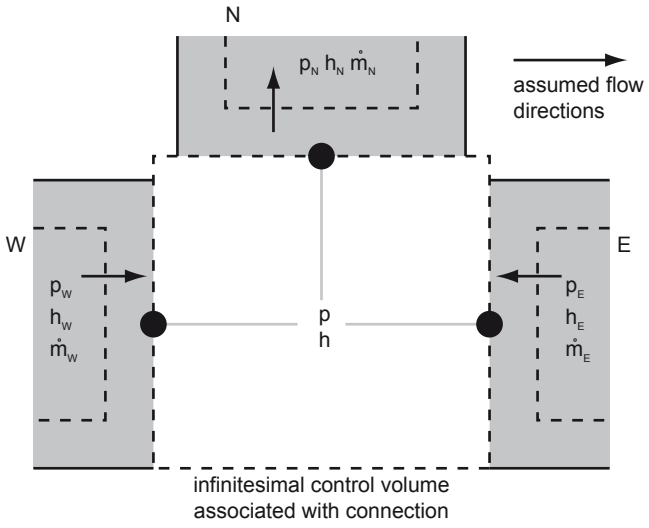


Figure 4.4: Details of three-way connection.

h. This is not what most users expect and it was found to be a continuous source of errors. From an object-oriented point of view, most users think of the connector as a representation of the adaptor of the real component. From that users tend to assume that the values in the connector correspond to the values one would obtain when conducting a measurement at the adaptor of the real component. But this is only true for connections of two components and for connector with a positive mass flow rate. In all other cases, the specific enthalpy is not the specific enthalpy of the fluid leaving the component but the mixing enthalpy h in the infinitesimal control volume. This is especially misleading when the components are positioned far from each other in the Modelica icon layer. The direct connection of three or more components involving refrigerant flows is not allowed in the new component model library and explicit junction elements have to be used in these cases.

A simple control volume is assumed to demonstrate the usage of `semiLinear()` in Modelica models. The control volume has two fluid ports called `portA` and `portB`. A refrigerant model from `TILFluids` is instantiated at the center of the cell. Code listing 4.2 shows the code that is required to couple the specific enthalpy of the refrigerant with the specific enthalpy in the port. An upwind differencing scheme (see for example Patankar, 1980) is used to set the specific enthalpy of the refrigerant object, i.e., the specific enthalpy at the port with negative mass flow rate is always equal to the specific enthalpy of the refrigerant.

```
model ControlVolume "Simple control volume with semiLinear()"
  FluidPort portA "Port A";
  FluidPort portB "Port B";

  TILFluids.Refrigerant refrigerant(refrigerantName="IfTLibrary.R744")
   "Refrigerant model";
equation
  portA.H_flow = semiLinear(portA.m_flow, portA.h, refrigerant.h);
  portB.H_flow = semiLinear(portB.m_flow, portB.h, refrigerant.h);
```

```
  ... // further code omitted
end ControlVolume;
```

Code Listing 4.2: Modelica code for simple control volume with `semiLinear()`.

The `Connectors` package contains different connector models for refrigerants, gases, and liquids. The main difference of these connector models is the icon that uniquely identifies the type of fluid flowing through the connector. There also are different connectors for arrays of connectors and for connectors with zero-flow that can be used to connect sensors from the `Sensors` package. The `FluidPort` contains the additional integer `index` that can be used to automatically determine the flow direction during online- or post-processing (see chapter 5).

The presented connector design is not the only way to formulate connectors for fluid systems. Another common approach is to provide both, the upstream and the downstream enthalpies, in the connectors as presented by Casella and Leva (2003, 2006) in their papers on modeling of thermo-hydraulic processes. Which connector design is superior depends on the kind of problem to be simulated and on the implementation of the `semiLinear()` function and is still the topic of an ongoing discussion in the Modelica Association.

4.3 Conservation Laws

The balance equations for energy, mass, and momentum are the starting point for the formulation of the system of equations describing a real system mathematically. Good introductions to the mathematical backgrounds are given in many fluid dynamics text book such as Patankar (1980), Versteeg and Malalasekera (1995), and Ferziger and Perić (2002). Formulations specific to the the field of component models for thermodynamic simulations can be found at Adiprasito (1998), Tegethoff (1999), Tummescheit (2002), and Pfafferott (2004). Many libraries formulate the conservation laws in a base class that is inherited in each component model. This tends to yield a very complex library structure that is hard to understand for developers as well as for users as explained in chapter 2. The new component model library does not define a base class like that but formulates the appropriate conservation laws in each component model. This yields a very simple and thus easy to understand structure where inheritance is only used to represent an *is-a*-relation as explained in section 2.3.3.

This section presents the formulation of the conservation laws that is used throughout all transient components in the `HVAC_p` package. The following section 4.4 shows how the conservation laws are implemented in the accumulator model. The cell models that are used as base elements for tubes and heat exchangers and that also implement the conservation laws are presented in section 4.7. The advantages of the specific formulation of the momentum balance that is based on a work by Lemke (2005) are explained in the last paragraphs of this section.

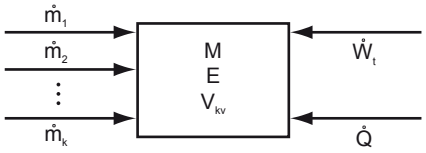


Figure 4.5: General control volume used for the formulation of the mass and energy balances.

All actually three-dimensional fluid flows are treated as one-dimensional flows within the scope of this thesis. This approach is applicable due to the distinct flow direction in all described components. The simplification of three-dimensional flows to one-dimensional flows is a very common technique in fluid modeling and is used by many authors (e.g., Tummescheit, 2002; Pfafferott, 2004; Casella et al., 2006; Casella and Leva, 2006). Figure 4.5 shows a control volume that is used for the formulation of the mass and energy balance where E is the energy in the control volume. The mass balance can be written as

$$\frac{dM}{dt} = \sum_k \dot{m}_k \tag{4.3}$$

where M is the total mass within the control volume and $\dot{m}_k$ are k mass flow rates entering and leaving the control volume. The first law of thermodynamics for an open system is (see Köhler, 2007a)

$$\frac{dE}{dt} = \frac{d}{dt}\int_{V_{kv}} \varrho \left(u + \frac{w^2}{2} + gz\right) dV = \sum_k \left[\dot{m}\left(h + \frac{w^2}{2} + gz\right)\right]_k + \dot{Q} + \dot{W}_t - p\frac{dV_{kv}}{dt} \tag{4.4}$$

where V_{kv} is the control volume, $\dot{Q}$ is the heat flow rate, and $\dot{W}_t$ is the shaft work which is called *technische Arbeit* in the German-speaking literature. Equation (4.4) can be simplified to

$$\frac{dU}{dt} = \sum_k \left[\dot{m}h\right]_k + \dot{Q} + \dot{W}_t - p\frac{dV_{kv}}{dt} \tag{4.5}$$

when neglecting the influences of the potential and the kinetic energy and assuming constant properties throughout the control volume. Equation (4.5) can be rewritten as follows using the relation $U = H - pV_{kv}$

$$\frac{dH}{dt} - p\frac{dV_{kv}}{dt} - V_{kv}\frac{dp}{dt} = \sum_k \left[\dot{m}h\right]_k + \dot{Q} + \dot{W}_t - p\frac{dV_{kv}}{dt} \tag{4.6}$$

This equation can be further simplified by taking into account

$$\frac{dH}{dt} = \frac{d(Mh)}{dt} = M\frac{dh}{dt} + h\frac{dM}{dt} = M\frac{dh}{dt} + h\sum_k \dot{m}_k \tag{4.7}$$

where equation (4.3) has been used to transform dM/dt. This yields the following formulation for the energy conservation

$$\frac{dh}{dt} = \frac{1}{M}\left\{\sum_k \left[\dot{m}_k(h_k - h)\right] + \dot{Q} + \dot{W}_t + V_{kv}\frac{dp}{dt}\right\} \tag{4.8}$$

The momentum equation is the most difficult equation to solve and different approaches can be found in the literature. For the derivation of the momentum balance, only two mass flow rates, $\dot{m}_{in}$ and $\dot{m}_{out}$, are considered as shown in figure 4.6.

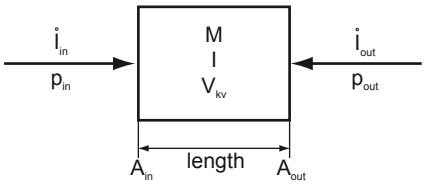


Figure 4.6: Control volume used for the formulation of the momentum balance.

Assuming that the two areas A_{in} and A_{out} are equal, the momentum equation can be written as (see Tummescheit, 2002; Pfafferott, 2004)

$$\frac{dI}{dt} = \dot{I}_{in} + \dot{I}_{out} + (p_{in} - p_{out})A - \Delta p_f A \tag{4.9}$$

where I is the momentum in the control volume, $\dot{I}_{in}$ and $\dot{I}_{out}$ are the momentum flows, and Δp_f is the friction pressure loss (see section 4.5 for details on friction pressure drop models). The momentum equation can be solved for the pressure change of the refrigerant in the control volume yielding

$$p_{out} - p_{in} = \Delta p = -\frac{1}{A}\frac{dI}{dt} + \frac{1}{A}\left(\dot{I}_{in} + \dot{I}_{out}\right) - \Delta p_f \tag{4.10}$$

The time derivative of the momentum in the control volume dI/dt is assumed to be zero which is reasonable as long as fast dynamic processes (e.g., sound propagation) are not considered (see Casella and Leva, 2006). The term can easily be added to the momentum balance used in the new component model library if dynamic processes are to be simulated for which this term is required. In many cases it is also possible to neglect the difference between the two momentum flows so that equation (4.10) can be written as

$$\Delta p = -\Delta p_f \tag{4.11}$$

Lemke (2005) showed that the time derivative of pressure can be treated as constant along the direction of flow

$$\frac{dp_{in}}{dt} = \frac{dp_{out}}{dt} \tag{4.12}$$

yielding a very efficient system of equations for each control volume since dp/dt is no longer an unknown in equation (4.8). It is important to note that the time derivative of pressure is not treated as constant over time but in space along the direction of flow.

The presented formulation of the conservation laws still requires a symbolic manipulation of the DAE system similar to an index reduction caused by the calculation rule for the mass in the control volume

$$M = \varrho V_{kv} \tag{4.13}$$

that is differentiated for equation (4.3) yielding

$$\frac{dM}{dt} = \frac{d(\varrho V_{kv})}{dt} = \varrho\frac{dV_{kv}}{dt} + V_{kv}\frac{d\varrho}{dt} \tag{4.14}$$

This yields three differential variables for each control volume assuming that the control volume itself does not change over time (i.e., $dV_{kv}/dt = 0$): pressure p, specific enthalpy h, and density ϱ. Only two of these three quantities are independent. The third quantity can be computed from the other two using an equation of state. In order to avoid the symbolic manipulation, the equation for the time derivative of density $d\varrho/dt$ is formulated explicitly as

$$\frac{d\varrho}{dt} = \left(\frac{\partial \varrho}{\partial p}\right)_h \frac{dp}{dt} + \left(\frac{\partial \varrho}{\partial h}\right)_p \frac{dh}{dt} \tag{4.15}$$

where the two partial derivatives are fluid properties that are computed in the refrigerant model from `TILFluids`. The equation (4.15) itself is formulated in the component models as shown in section 4.4 for the accumulator and in section 4.7.1 for the refrigerant cell. A case differentiation between a one-phase state and a two-phase state is required to compute the two partial derivatives of $\varrho(p, h)$ in equation (4.15). For the one-phase state, the partial derivatives as functions of β, κ, and c_p can be computed using Bridgman's tables (see Bejan, 1988) yielding

$$\left(\frac{\partial \varrho}{\partial h}\right)_p = -\frac{\beta \varrho}{c_p} \tag{4.16}$$

$$\left(\frac{\partial \varrho}{\partial p}\right)_h = \frac{-T\beta^2 + \beta + \kappa \varrho c_p}{c_p} \tag{4.17}$$

where β is the isobaric coefficient of expansion, κ is the isothermal compressibility, and c_p is the specific heat capacity at constant pressure which can be determined from medium properties

$$\beta = -\frac{1}{\varrho}\left(\frac{\partial \varrho}{\partial T}\right)_p, \quad \kappa = \frac{1}{\varrho}\left(\frac{\partial \varrho}{\partial p}\right)_T, \quad \text{and} \quad c_p = \left(\frac{\partial u}{\partial T}\right)_p \tag{4.18}$$

The equation for the two-phase state are slightly more complicated and are derived in appendix D.

Lemke (2005) was the first to describe the simplifying assumption from equation (4.12). He uses this formulation in the basic cell of his heat exchanger model. The formulation is extended in this thesis to all component models and used in cycle computations with very satisfying results.

4.4 Accumulator Model

The accumulator model is a good example for a component model implementing the conservation laws presented in the last section. The following set of equations describes a simple

transient accumulator with constant vapor properties at the outlet

$$M = V\varrho \tag{4.19a}$$

$$\frac{d\varrho}{dt} = \left(\frac{\partial\varrho}{\partial h}\right)_p \frac{dh}{dt} + \left(\frac{\partial\varrho}{\partial p}\right)_h \frac{dp}{dt} \tag{4.19b}$$

$$V\frac{d\varrho}{dt} = \dot{m}_{in} + \dot{m}_{out} \tag{4.19c}$$

$$p_{out} = p_{in} \tag{4.19d}$$

$$\frac{dh}{dt} = \frac{1}{M}\left(\dot{m}_{in}(h_{in} - h) + \dot{m}_{out}(h_{out} - h) + V\frac{dp}{dt}\right) \tag{4.19e}$$

$$h_{out} = f(x_{in}, h, \dots) \tag{4.19f}$$

The mass, momentum, and energy balance are given in equation (4.19c), (4.19d), and (4.19e) respectively. Equation (4.19b) describes the time derivative of density within the accumulator. Note that the time derivative of pressure is provided as an input to the model according to the simplification discussed in the previous section. The two partial derivatives of density with respect to pressure and specific enthalpy are computed in the medium model. The outlet enthalpy depends on the quality of the entering fluid, the specific enthalpy in the accumulator, and various other factors such as the filling level as stated in equation (4.19f). In the very simple accumulator model presented in code listing 4.3, a constant outlet condition of $x_{out} = 1$ is assumed.

```
import SI = Modelica.SIunits;

model Accumulator "Simple accumulator model"
  FluidPort inlet "Inlet port";
  FluidPort outlet "Outlet port";

  parameter SI.Volume volume=1e-3 "Volume";
  SI.Mass mass "Mass";

  SI.SpecificEnthalpy hOutlet "Outlet specific enthalpy";
  Real drhodt "Time derivative of density";
  Real dpdt "Time derivative of pressure";

  TILFluids.Refrigerant refrigerant(refrigerantName="IfTLibrary.R744")
   "Refrigerant model";
equation
  refrigerant.p = (inlet.p + outlet.p)/2.0;

  inlet.H_flow = semiLinear(inlet.m_flow, inlet.h, refrigerant.h);
  outlet.H_flow = semiLinear(outlet.m_flow, outlet.h, hOutlet);

  mass = volume*refrigerant.d;
  drhodt = refrigerant.drhodh*der(refrigerant.h) + refrigerant.drhodp*dpdt;
  hOutlet = refrigerant.sat.hv; // to be replaced in advanced models with more realistic
                                // relations based on measurements
```

```
  volume*drhodt = inlet.m_flow + outlet.m_flow "Mass balance";
  inlet.p - outlet.p = 0 "Momentum balance";
  der(refrigerant.h) = 1/mass*(inlet.m_flow*(inlet.h - refrigerant.h) +
   outlet.m_flow*(outlet.h - refrigerant.h) + volume*dpdt) "Energy balance";

  ... // further code omitted
end Accumulator;
```

Code Listing 4.3: Code for `Accumulator`.

This assumption can be replaced by more realistic descriptions based on measurements from real accumulators. Raiser (2005) describes a dynamic model for an accumulator taking into account the density distribution and a possible bypass. Another dynamic accumulator model is described by Strupp et al. (2007) and Bockholt et al. (2008) and is also available in the `Accumulator` package. This advanced model uses the filling level of the accumulator to specify different operating regions and provides specific functions for the outlet properties depending on the current operating region.

4.5 Heat Transfer and Pressure Drop Models

This section describes the heat transfer and pressure drop models implemented in the new component model library that are used in different control volumes such as the cells presented in section 4.7. The pressure drop models compute the friction pressure loss Δp_f in equation (4.11) depending on the flow condition and on the geometry. The most simple pressure drop models assume the friction pressure drop Δp_f to be constant.

The heat flow rate $\dot{Q}$ in equation (4.8) is computed from

$$\dot{Q} = \alpha A(T_w - T) \tag{4.20}$$

where α is the heat transfer coefficient, A is the heat transfer area, T_w is the wall temperature, and T the temperature in the control volume. The heat transfer models compute the coefficient of heat transfer α depending on the flow condition and on the geometry. The most simple heat transfer models assume a constant coefficient of heat transfer α.

Some advanced models for both, heat transfer and pressure drop, for refrigerants and gases are explained in the following sections. Many models are only valid within certain regions. Smooth transition functions that are described in section 4.6 are provided in the `Utilities` package. These functions are used for a smooth transition between different regions.

4.5.1 Single-Phase Refrigerant Flows

The heat transfer coefficient α can be computed from

$$\alpha = \frac{Nu\ \lambda}{d_h} \tag{4.21}$$

where Nu is the Nusselt number, λ is the thermal conductivity, and d_h is the hydraulic diameter. The hydraulic diameter for a channel is

$$d_h = 4\frac{A}{U} \tag{4.22}$$

where A is the cross sectional area and U is the perimeter. This definition can be replaced by more advanced definitions in the case of complex three-dimensional geometries (see Wagner, 2001). All following relations are given for tubes with circular cross sectional area with a single phase fluid flow. The Nusselt number can be computed depending on the flow regime. For laminar flow, a constant Nusselt number

$$Nu = 3.6568 \tag{4.23}$$

can be used. Gnielinski (see Baehr and Stephan, 2004) gives a correlation for the Nusselt number that is valid from the laminar flow regime ($Re < 2300$) to Reynolds numbers up to $5 \cdot 10^6$ (for $0.5 \leq Pr \leq 2000$)

$$Nu = \frac{(\zeta/8)(Re - 1000)Pr}{1 + 12.7\sqrt{\zeta/8}(Pr^{2/3} - 1)} \tag{4.24}$$

where Pr is the Prandtl number, and ζ is the friction factor which can be computed from

$$\zeta = \frac{1}{(0.79 \ln Re - 1.64)^2} \tag{4.25}$$

For higher Reynolds numbers ($Re \geq 10^4$) the correlation by Dittus-Bölter can be used

$$Nu = 0.023\ Re^{4/5}\ Pr^{1/3} \tag{4.26}$$

to compute the Nusselt number.

For the description of the pressure drop, the pipe friction factor λ is used as proportionality factor

$$\Delta p_f = \lambda\ \frac{L}{d}\ \frac{\varrho}{2}\ w_m|w_m| \tag{4.27}$$

where L is the length, d is the diameter, and w_m is the average velocity. For complex geometries, the total proportionality factor $\lambda \cdot L/d$ is usually determined from experiments. This factor is called friction factor ζ yielding the following relation for the pressure drop

$$\Delta p_f = \zeta \frac{\varrho}{2}\ w|w| \tag{4.28}$$

For tubes

$$\zeta = \lambda \frac{L}{d} \tag{4.29}$$

The pipe friction factor for laminar flow is given by

$$\lambda = \frac{64}{Re} \tag{4.30}$$

The pipe friction factor for higher Reynolds numbers ($2300 \leq Re \leq 10^7$) is given by the relation by Konakov (see Wagner, 2001) that is valid for smooth pipes

$$\lambda = \frac{1}{(1.80 \log Re - 1.5)^2} \tag{4.31}$$

4.5.2 Two-Phase Refrigerant Flows

The condensation in tubes can be described with a simple model by Shah (1979) that is also referenced by Baehr and Stephan (2004) and that describes the turbulent film condensation inside pipes. This model computes the Nusselt number as

$$Nu = 0.023\ Re^{0.8}\ Pr_l^{0.4}\ \left((1-x^*)^{0.8} + \frac{3.8(1-x^*)^{0.04}\ x^{*0.76}}{p_{red}^{0.38}}\right) \tag{4.32}$$

where

$$Nu = \frac{\alpha\ d_h}{\lambda_l}, \quad Re = \frac{\dot{m}\ d_h}{\varrho_l A\ \nu_l}, \quad \text{and} \quad p_{red} = \frac{p}{p_c} \tag{4.33}$$

The flow steam quality x^* can be computed from

$$x^* = \frac{\dot{m}_g}{\dot{m}} \tag{4.34}$$

where $\dot{m}_g$ is the mass flow rate of the gas phase. A slip factor s is introduced to describe the velocity differences between the liquid and the gas phase

$$s = \frac{w_g}{w_l} = \frac{x^*}{1-x^*}\frac{1-\varepsilon}{\varepsilon}\frac{\varrho_l}{\varrho_g} \tag{4.35}$$

where ε is the void fraction defined as

$$\varepsilon = \frac{V_g}{V} \tag{4.36}$$

Note that the flow steam quality x^* is equal to the quality $x = m_g/m$ if the velocities of the two phases are equal, i.e., $s = 1$. The void fraction for this simple case called homogeneous flow can be simplified to

$$\varepsilon = \left(\frac{\varrho_g}{\varrho_l}\frac{1-x^*}{x^*} + 1\right)^{-1} \tag{4.37}$$

For all other cases, many empirical relations are given in the literature to compute the void fraction ε depending on the flow steam quality x^*, the saturation densities ϱ_l and ϱ_g, the saturation viscosities μ_l and μ_g, and geometric parameters. Woldesemayat and Ghajar (2007) give a very good overview of various empirical relations and compare them to results from measurements. All currently available models in the new component model library assume non-slip flow by default but allow for the explicit formulation of an empirical correlation for ε. First tests with the correlation by Premoli et al. (1970) that is also suggested by Heinrich and Berthold (2006) showed promising results but requires further testing that goes beyond the scope of this thesis.

For evaporation, two different models are used depending on the Froude number Fr that compares the inertial and the gravitational forces and is defined as

$$Fr = \frac{\dot{m}}{A^2\varrho_l^2 g d_h} \tag{4.38}$$

For $Fr < 0.04$, Baehr and Stephan (2004) use a correlation given by Shah (1976) to compute the heat transfer coefficient α

$$\alpha = 3.9\ Fr^{0.24}\left(\frac{x^*}{1-x^*}\right)^{0.64}\left(\frac{\varrho_l}{\varrho_g}\right)^{0.4}\alpha_K \tag{4.39}$$

where

$$\alpha_K = \frac{\lambda_l}{d_h}\, 0.023\, Re^{0.8} Pr^{0.4}, \quad \text{and} \quad Re = \frac{\dot{m}(1-x^*)d_h}{\varrho_l A\, \nu_l} \tag{4.40}$$

For $Fr > 0.04$, an approach by Chen (1966) is proposed by Baehr and Stephan (2004) in which a heat transfer coefficient for bulk boiling α_B and a heat transfer coefficient for convective boiling α_K are used to determine the total coefficient of heat transfer according to

$$\alpha = S\alpha_B + F\alpha_K \tag{4.41}$$

The factors S and F can be computed from

$$F = 1 + 2.4 \cdot 10^4\, Bo^{1.16} + 1.37\, X_{tt}^{-0.86} \tag{4.42}$$

$$S = \left(1 + 1.15 \cdot 10^{-6} F^2 Re^{1.17}\right)^{-1} \tag{4.43}$$

with

$$Re = \frac{\dot{m}(1-x^*)d_h}{\varrho_l A\, \nu_l}, \quad Bo = \frac{\dot{q}A}{\dot{m}(h_g - h_l)}, \quad \text{and} \quad X_{tt} = \left(\frac{1-x^*}{x^*}\right)^{0.9} \left(\frac{\nu_l \varrho_l}{\nu_g \varrho_g}\right)^{0.1} \left(\frac{\varrho_g}{\varrho_l}\right)^{0.5} \tag{4.44}$$

The heat transfer coefficient α_K can be computed using a relation for turbulent single-phase flow as presented in section 4.5.2. α_B can be computed from a relation for evaporation in a free flow. This relation is based on a known heat transfer coefficient α_0 and the corresponding heat flux density $\dot{q}_0$. The heat transfer coefficient for other pressures and heat flux densities can be determined from

$$\alpha_B = \alpha_0 F(p_{red}) \left(\frac{\dot{q}}{\dot{q}_0}\right)^n \tag{4.45}$$

Stephan (1988) gives the following relation for the pressure correction $F(p_{red})$ and the coefficient n that is valid for organic fluids and for ammonia

$$F(p_{red}) = 2.1\, p_{red}^{0.27} + \left(4.4 + \frac{1.8}{1 - p_{red}}\right) p_{red}, \quad \text{and} \quad n = 0.9 - 0.3 p_{red}^{0.3} \tag{4.46}$$

Reference values for α_0 are available for many fluids. Some of them are stored in `TILFluids` and can be obtained via a function call. Smooth transition functions (see section 4.6) are used to switch between single-phase and two-phase flows depending on the quality x. The sign of the heat flux density $\dot{q}$ is used to switch between evaporation and condensation models in the case of two-phase flows.

The pressure drop in two-phase flows consists of two different parts, a static and a dynamic pressure, and can be computed from (see Baehr and Stephan, 2004)

$$\Delta p = -\tau_0 \frac{U}{A} - \frac{1}{A}\frac{d(\dot{m}w)}{dx} \tag{4.47}$$

where τ_0 is the wall shear stress, A the cross-sectional area, U the perimeter of the channel, and x the coordinate in flow direction. The momentum flow $\dot{m}w$ is composed of the momentum flow of the gas and of the liquid respectively

$$\dot{m}w = \dot{m}_g w_g + \dot{m}_l w_l \tag{4.48}$$

Mayinger (1982) uses a two-phase multiplier R to compute the static pressure of a two-phase flow according to the relation for single-phase flow

$$\frac{U}{A}\tau_0 = R\frac{\zeta_l}{2d_h\varrho_l}\frac{\dot{m}|\dot{m}|}{A^2} \tag{4.49}$$

where the two-phase multiplier R can be computed from

$$R = (1-x^*)^2 + x^{*2}\left(\frac{\varrho_l\zeta_g}{\varrho_g\zeta_l}\right) + 3.43x^{*0.24}(1-x^*)^{0.24}\cdot \left(\frac{\varrho_l}{\varrho_g}\right)^{0.8}\left(\frac{\mu_g}{\mu_l}\right)^{0.22}\left(1-\frac{\mu_g}{\mu_l}\right)^{0.89} Fr^{-0.047}\,We^{-0.0334} \tag{4.50}$$

where the Froude number Fr and the Weber number We are defined as

$$Fr = \frac{\dot{m}^2}{gd_h\varrho^2A^2}, \quad \text{and} \quad We = \frac{\dot{m}^2d_h}{\varrho\sigma A^2} \tag{4.51}$$

The friction factors ζ_l and ζ_g of the liquid and the gas flow depend on the Reynolds numbers Re_l and Re_g. The Reynolds numbers are used to switch between a laminar and a turbulent approach according to

$$\zeta_{l,g} = \begin{cases} \dfrac{64}{Re_{l,g}} & Re_{l,g} \le 1055 \\ \text{and} & \\ \left(0.86859\ln\left(\dfrac{Re_{l,g}}{\ln(Re_{l,g})-3.8215}\right)\right)^{-2} & Re_{l,g} > 1055 \end{cases} \tag{4.52}$$

where

$$Re_{l,g} = \frac{\dot{m}d_h}{A\mu_{l,g}} \tag{4.53}$$

Again, smooth transition functions (see section 4.6) are used to switch between the single-phase and two-phase and between the different flow regimes.

4.5.3 Gas Flows

The most advanced heat transfer and pressure drop correlations in gas cells are correlations based on the work of von Haaf (1988) that were also used by Tegethoff (1999) and Schmidt (2002). This approach uses the following hydraulic diameter to compute the Reynolds and the Nusselt numbers

$$d_h = \frac{4V\Psi}{A_a} \tag{4.54}$$

where Ψ is the void fraction, V is the total volume of the heat exchanger, and A_a is the total surface area of the heat exchanger. The following relations can be derived from geometrical considerations

$$V = l_{lam}z_ls_lz_qs_q \tag{4.55}$$

$$\Psi = 1 - \frac{\delta_R}{t_R} - \frac{\pi d_a^2(t_R-\delta_R)}{4s_qs_tt_R} \tag{4.56}$$

$$A_a = \left[2s_qs_l + \pi d_a\left(t_R - \delta_R - \frac{d_a}{2}\right)\right]\frac{l_{lam}}{t_R}z_qz_l \tag{4.57}$$

where l_{lam} is the length of the finned tube, and z_l and z_q are the number of serial and parallel tubes respectively. All other geometric parameters are shown in figure 4.7.

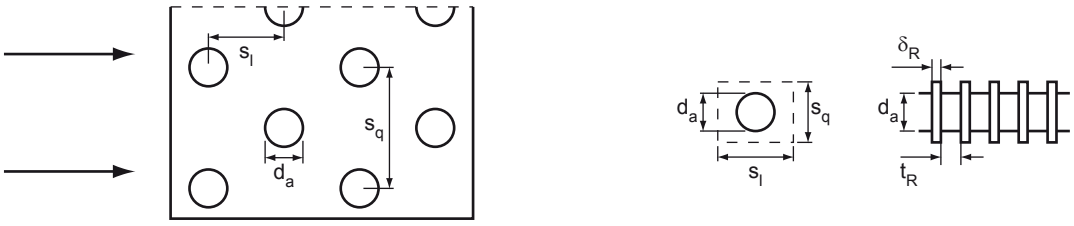


Figure 4.7: Fin geometry.

The Reynolds number is computed using the average air velocity $w_{l,m}$ determined by

$$w_{l,m} = \frac{v_l}{\Psi} \tag{4.58}$$

where v_l is the air velocity in the clear cross section. Haaf uses the experimental results from six different publications to derive the following correlation for the heat transfer correlation

$$Nu = 0.31\ Re^{5/8} Pr^{1/3} \left(\frac{d_h}{s_l}\right)^{1/3} \tag{4.59}$$

The coefficient of heat transfer α is computed from this equation using equation (4.21).

Von Haaf uses the friction factor ξ as dimensionless parameter for the pressure drop correlation

$$\Delta p_f = \frac{s_l}{d_h} \xi \frac{\varrho}{2} \frac{w_l}{\Psi^2} |w_l| \tag{4.60}$$

An empirical approach is used to determine ξ

$$\xi = 10.5\ Re^{-1/3} \left(\frac{d_h}{s_l}\right)^{3/5} \tag{4.61}$$

4.5.4 Solids

Figure 4.8 shows a schematic drawing of the temperature distribution in a solid wall. The heat flow rate $\dot{Q}$ can be computed from

$$\dot{Q} = \frac{T_1 - T_2}{R} \tag{4.62}$$

where T_1 and T_2 are the surface temperatures at each side and R is the thermal resistance.

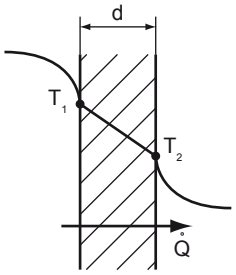


Figure 4.8: Heat flow through solid wall.

The thermal resistance R depends on the geometry of the wall. For a planar wall with thickness d, total area A, and thermal conductivity λ, the thermal resistance R is

$$R = \frac{d}{A\lambda} \tag{4.63}$$

The thermal resistance for concentric tube walls with an inner radius r_1, an outer radius r_2, and a total length L can be computed from (see Polifke and Kopitz, 2005)

$$R = \frac{\ln\left(\frac{r_2}{r_1}\right)}{2\pi\lambda L} \tag{4.64}$$

4.6 Smooth Transition Functions

Transition functions are required in many component models to switch smoothly and continuously between different functions. A good example are heat transfer and pressure drop correlations that are usually only valid within certain regions. The continuous transition between those regions is a typical problem in numerical simulations. Transition functions are provided in the `Utilities` package to allow for a smooth transition between different functions with a variable number of smooth derivatives. These functions $T(x)$ can be used in the following way to switch between the two functions $f(x)$ and $g(x)$

$$z(x) = T(x) \cdot f(x) + (1 - T(x)) \cdot g(x) \tag{4.65}$$

where the transition function $T(x)$ is defined as

$$T(x) = \begin{cases} 1 & x \leq x_t - \frac{\Delta x}{2} \\ t(x) \quad \text{for} & x_t - \frac{\Delta x}{2} < x < x_t + \frac{\Delta x}{2} \\ 0 & x \geq x_t + \frac{\Delta x}{2} \end{cases} \tag{4.66}$$

x_t being the transition point, Δx the transition length, and $t(x)$ a function with

$$t(x_t - \tfrac{\Delta x}{2}) = 1 \quad \text{and} \quad t(x_t + \tfrac{\Delta x}{2}) = 0 \tag{4.67}$$

In order that $T(x)$ is smooth, the following restrictions apply for $t(x)$

$$\left.\frac{dt(x)}{dx}\right|_{x=x_t-\frac{\Delta x}{2}} = 0 \quad \text{and} \quad \left.\frac{dt(x)}{dx}\right|_{x=x_t+\frac{\Delta x}{2}} = 0 \tag{4.68}$$

The simplest way to find a suitable function is to start with the restrictions stated in equation (4.68). The fundamental trigonometric function

$$\frac{dt(\varphi)}{d\varphi} = a\cos(\varphi) \tag{4.69}$$

where a is a scaling factor and φ the phase defined by

$$\varphi = \frac{x - x_t}{\Delta x}\,\pi \tag{4.70}$$

fulfills these restrictions. This yields the following function for $t(\varphi)$

$$t(\varphi) = a\sin(\varphi) + b \tag{4.71}$$

The two parameters a and b can be computed from equation (4.67). From equation (4.69) follows that the first derivative of the transition function $T(x)$ defined as

$$\frac{dT(x)}{dx} = \begin{cases} 0 & x \leq x_t - \frac{\Delta x}{2} \\ \frac{dt(x)}{dx} & \text{for} \quad x_t - \frac{\Delta x}{2} < x < x_t + \frac{\Delta x}{2} \\ 0 & x \geq x_t + \frac{\Delta x}{2} \end{cases} \tag{4.72}$$

is not smooth at $x = x_t - \frac{\Delta x}{2}$ and $x = x_t + \frac{\Delta x}{2}$. This discontinuity can have negative effects on the solution process because the solver has to handle functions with discontinuous first derivatives. Equation (4.69) can be generalized to

$$\frac{dt(\varphi)}{d\varphi} = a\cos^n(\varphi) \tag{4.73}$$

where n is a positive integer yielding the following function for $t(\varphi)$

$$t(\varphi) = a\frac{cos^{n-1}(\varphi)\sin(\varphi)}{n} + \frac{n-1}{n}\int \cos^{n-2}(\varphi)d\varphi + b \tag{4.74}$$

The two parameters a and b can again be computed from equation (4.67). The resulting function $t(\varphi)$ is $(n-1)$th order continuous. Table 4.1 shows the functions and values for a and b for $t(\varphi)$ for $n = 1, 2, 3, 4$. Figure 4.9 and 4.10 show the function $T(x)$ and its derivative $dT(x)/dx$ respectively for $n = \{1, 2, 3\}$.

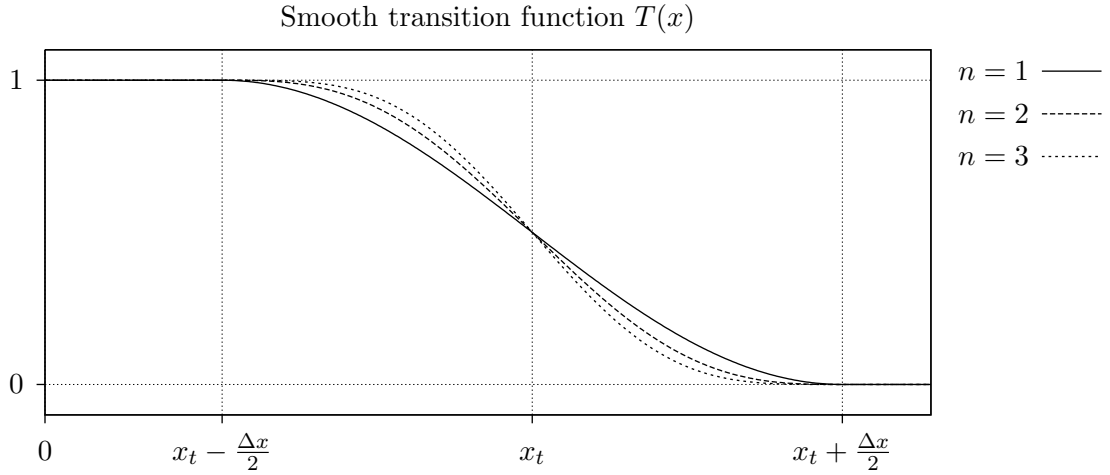


Figure 4.9: Smooth transition functions $T(x)$ for $n = \{1, 2, 3\}$.

n	$\mathbf{t}(\varphi)$	a	b
1	$a\sin(\varphi)+b$	$-\frac{1}{2}$	$\frac{1}{2}$
2	$a\left(\frac{1}{2}\cos(\varphi)\sin(\varphi)+\frac{1}{2}\varphi\right)+b$	$-\frac{2}{\pi}$	$\frac{1}{2}$
3	$a\left(\frac{1}{3}\cos(\varphi)^2\sin(\varphi)+\frac{2}{3}\sin(\varphi)\right)+b$	$-\frac{3}{4}$	$\frac{1}{2}$
4	$a\left(\frac{1}{4}\cos(\varphi)^3\sin(\varphi)+\frac{3}{8}\cos(\varphi)\sin(\varphi)+\frac{3}{8}\varphi\right)+b$	$-\frac{8}{3\pi}$	$\frac{1}{2}$

Table 4.1: Functions $t(\varphi)$ used in smooth transition functions $T(x)$ provided in the `Utilities` package of the new component model library.

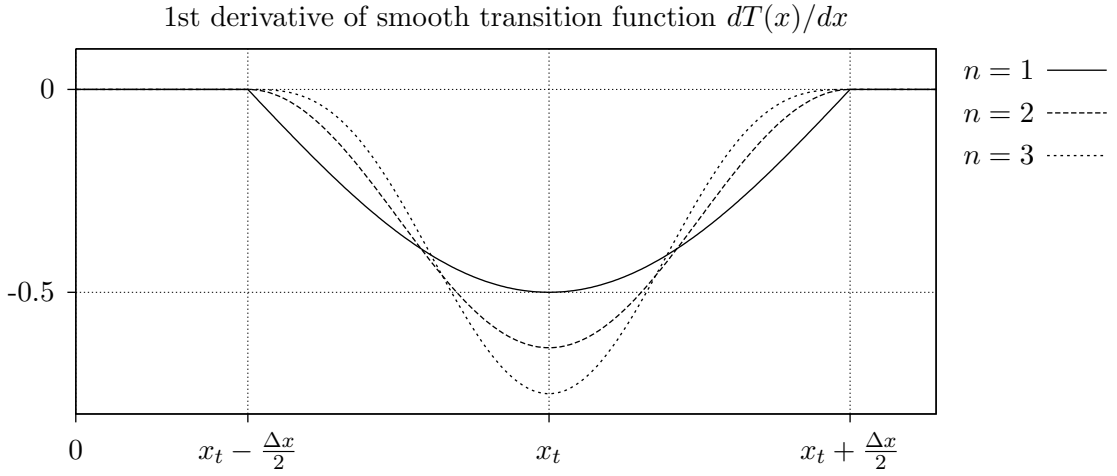


Figure 4.10: Derivatives of smooth transition function $dT(x)/dx$ for $n = \{1, 2, 3\}$.

A smooth function $t(x)$ that is $(n-1)$th order continuous is only one requirement to obtain a function $z(x)$ that is $(n-1)$th order continuous. The other two requirements concern the two functions $f(x)$ and $g(x)$. Differentiating equation (4.65) yields

$$\frac{dz(x)}{dx} = \frac{dT(x)}{dx}f(x) + T(x)\frac{df(x)}{dx} - \frac{dT(x)}{dx}g(x) + (1-T(x))\frac{dg(x)}{dx} \tag{4.75}$$

from which follows that $f(x)$ has to be $(n-1)$th order continuous for $x < x_t + \frac{\Delta x}{2}$ and $g(x)$ has to be $(n-1)$th order continuous for $x > x_t - \frac{\Delta x}{2}$ in order that $z(x)$ is $(n-1)$th order continuous.

A simple example is used to demonstrate how the smooth transition function is used. A function $g(x)$ is defined with a discontinuity at $x = 2$ according to

$$z^*(x) = \begin{cases} x & x \leq 2 \\ \text{and} & \\ x^2+1 & x > 2 \end{cases} \tag{4.76}$$

This function is smoothed using the transition function from equation (4.65) which yields

$$z(x) = T(x) \cdot x + (1 - T(x)) \cdot \left(x^2 + 1\right) \tag{4.77}$$

The two functions x and $x^2 + 1$ and the smooth transition function $z(x)$ for $n = 1, 2$ are shown in figure 4.11. The transition length is chosen to be $\Delta x = 2$. Figure 4.12 shows the corresponding derivatives. Note the discontinuity in $dz(x)/dx, n = 1$ at $x = 3$.

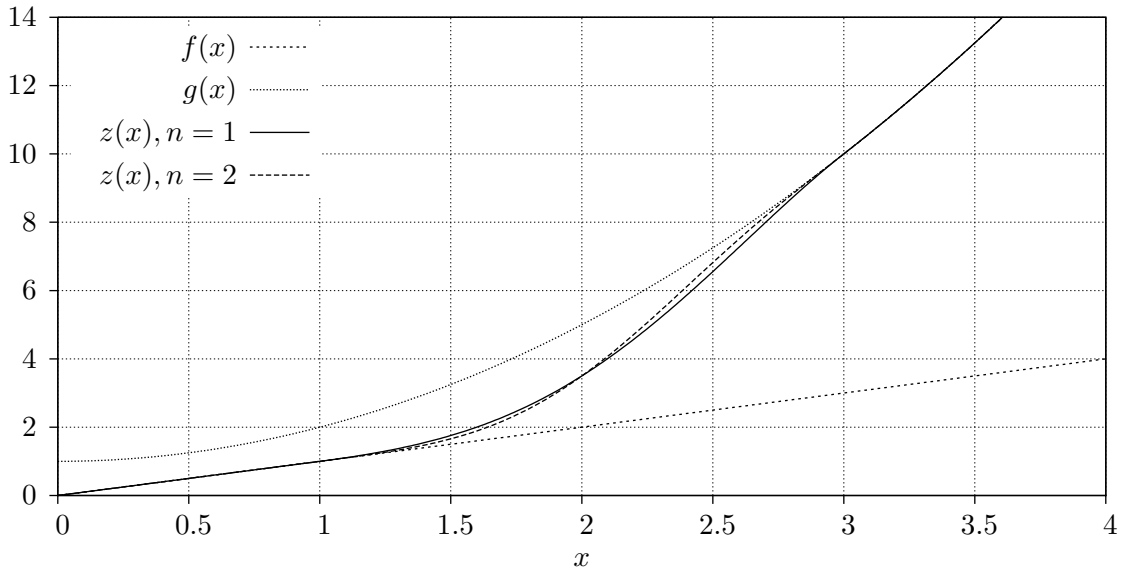


Figure 4.11: Example for smooth transition between two functions.

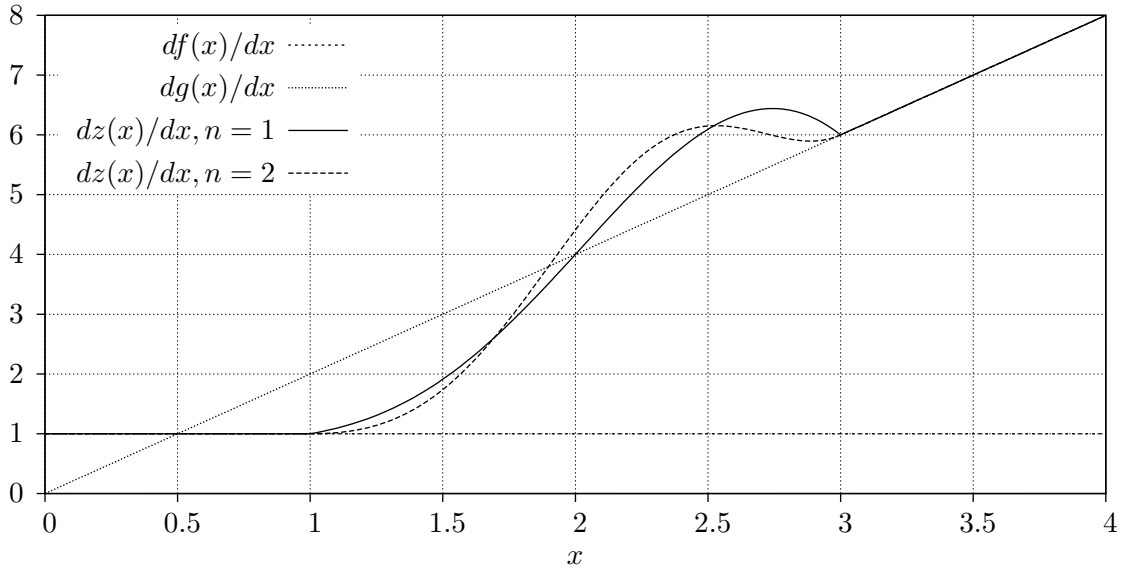


Figure 4.12: Example for derivative of smooth transition between two functions.

4.7 Cells for Refrigerants, Liquids, Gases, and Solids

Cells are the base elements that are used to assemble more complex structures such as tubes and heat exchangers in the new component model library. This section presents the four most important cells which are the `RefrigerantCell`, the `GasCell`, the `LiquidCell`, and the `WallCell`. A `MoistAirCell` that handles moist air including condensation and evaporation is also available but its description goes beyond the scope of this work. Having different cell models to handle refrigerants, gases, liquids, and moist air seems to be an overhead due to the duplication of the balance equations and is avoided in many other Modelica libraries by combining all these models into one base model as already discussed in chapter 3 for the fluid property library `Modelica.Media`. The major drawback of this *one-can-fit-all* solution is the high complexity of the resulting component model library due to the fact that the same model is used for such different media as ideal gases, simple liquids, and two-phase fluids. The cells in the new component model library were instead designed based on the design rules presented in chapter 2 and the medium models presented in section 3.5.

4.7.1 Refrigerant Cell

The `RefrigerantCell` is the base element for tubes and many heat exchanger models describing a finite volume of a discretized model. Each refrigerant cell contains a refrigerant object of type `Refrigerant` from `TILFluids` representing the fluid properties in the control volume. An upwind scheme is used for the discretization, i.e., the fluid properties at the outlet of the cell are identical with the fluid properties of the refrigerant model. The conservation laws described in section 4.3 are formulated in each refrigerant cell yielding the following set of equations

$$M = V\varrho \tag{4.78a}$$

$$\frac{d\varrho}{dt} = \left(\frac{\partial\varrho}{\partial h}\right)_p \frac{dh}{dt} + \left(\frac{\partial\varrho}{\partial p}\right)_h \frac{dp}{dt} \tag{4.78b}$$

$$V\frac{d\varrho}{dt} = \dot{m}_{in} + \dot{m}_{out} \tag{4.78c}$$

$$p_{out} = p_{in} - \Delta p_f \tag{4.78d}$$

$$\frac{dh}{dt} = \frac{1}{M}\left(\dot{m}_{in}(h_{in} - h) + \dot{m}_{out}(h_{out} - h) + \dot{Q} + V\frac{dp}{dt}\right) \tag{4.78e}$$

$$h = h_{out} \tag{4.78f}$$

$$p = \frac{p_{in} + p_{out}}{2} \tag{4.78g}$$

Note that the acceleration term is skipped in equation (4.78e) for simplicity but that it can be reintroduced if needed. The friction pressure drop Δp_f in equation (4.78d) is computed in the pressure drop model. For the heat transfer, an additional equation is formulated in the refrigerant cell

$$\dot{Q} = \alpha A(T_w - T) \tag{4.79}$$

where T_w is the wall temperature and T is the temperature of the refrigerant. The coefficient of heat transfer αA is computed in the heat transfer model.

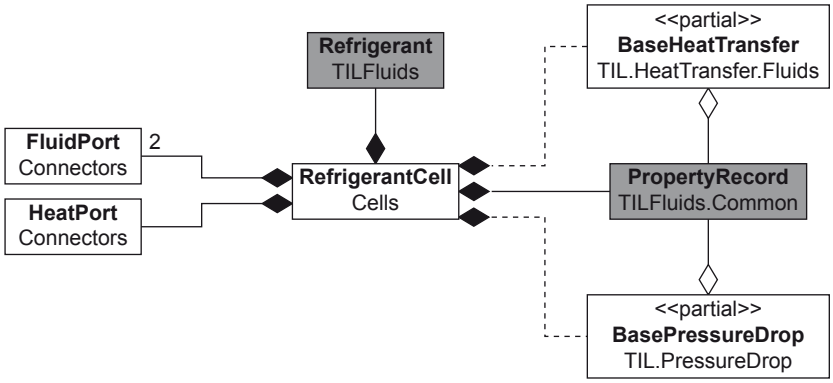


Figure 4.13: UML class diagram of `RefrigerantCell`. The two shown partial base models for heat transfer and pressure drop correlations are replaced when instantiating the refrigerant cell by one of the models presented in section 4.5.

Figure 4.13 shows the class diagram of a refrigerant cell. The gray models are models from `TILFluids`. The `PropertyRecord` provides a standardized interface to fluid properties independent from the type of the model itself (e.g., `Refrigerant`, `Gas`). Aggregation (see section 2.3.2) is used to make the fluid properties available in the pressure drop and heat transfer models. Two fluid ports and a heat port define the interfaces of the refrigerant cell. To simplify matters, the class diagram does not show a couple of additional variables made available in the heat transfer and pressure drop models using aggregation which are listed in table 4.2.

Variable	Unit	Description
`mdotHydraulic`	kg/s	Hydraulic mass flow rate used to compute the pressure drop
`qdotHydraulic`	W	Hydraulic heat flow rate used to compute the heat transfer
`wallTemperature`	K	Wall temperature used to compute the heat transfer

Table 4.2: Additional variables in cells that are provided to the heat transfer and pressure drop models using aggregation as presented in section 2.3.2.

The most advanced heat transfer and pressure drop models for refrigerant cells combine the correlations described in sections 4.5.1 and 4.5.2. Table 4.3 shows the heat transfer correlations implemented in the `AdvancedHeatTransfer` model. Smooth transition functions are used to switch between the different correlations.

		$\dot{\mathbf{q}} \geq \mathbf{0}$	$\dot{\mathbf{q}} < \mathbf{0}$
Single-Phase Flow	$Re \leq$ **2300**	constant Nusselt number (4.23)	
	2300 $< Re \leq$ **1e4**	Gnielinski (4.24)	
	$Re >$ **1e4**	Dittus-Bölter (4.26)	
Two-Phase Flow	$Fr \leq$ **0.04**	Shah (4.39)	Shah (4.32)
	$Fr >$ **0.04**	Chen (4.41)	

Table 4.3: Heat transfer correlations implemented in `AdvancedHeatTransfer` model.

Table 4.4 shows the pressure drop correlations implemented in the `AdvancedPressureDrop` model. Smooth transition functions are used to switch between the different correlations. All correlations work in both flow directions, $\dot{m} \geq 0$ and $\dot{m} < 0$.

Single-Phase Flow	$Re \leq \mathbf{2300}$	laminar flow (4.30)
	$Re > \mathbf{2300}$	turbulent flow (4.31)
Two-Phase Flow	$Re_{\mathbf{l,g}} \leq \mathbf{1055}$	laminar flow (4.49) and (4.52)
	$Re_{\mathbf{l,g}} > \mathbf{1055}$	turbulent flow (4.49) and (4.52)

Table 4.4: Pressure drop correlations implemented in `AdvancedPressureDrop` model.

The structure of the cell is kept as simple as possible with no inheritance following the design rules presented in chapter 2. The flat structure makes it very straightforward for users as well as for developers to understand the structure of the cell and to find potential errors. Other libraries such as the `AirConditioning` library use structures that are much more complicated and therefore harder to understand for developers as well as for users as discussed in section 2.7. Figure 2.12 for example shows the inheritance relations for the refrigerant cell used in heat exchanger models in the `AirConditioning` library. The inheritance tree is five levels deep and uses multiple inheritance in three places making it very hard to understand especially if no tools for an automated object-oriented analysis are available.

4.7.2 Gas Cell

Figure 4.14 shows the class diagram of a gas cell. Each gas cell contains a gas model `Gas` from `TILFluids` representing the fluid properties in the control volume. An upwind scheme is used for the discretization. The conservation laws are formulated steady-state in the control volume yielding the following set of equations

$$0 = \dot{m}_{in} + \dot{m}_{out} \tag{4.80a}$$

$$p_{out} = p_{in} - \Delta p_f \tag{4.80b}$$

$$0 = \dot{m}_{in} h_{in} + \dot{m}_{out} h_{out} + \dot{Q} \tag{4.80c}$$

$$h = h_{out} \tag{4.80d}$$

$$p = \frac{p_{in} + p_{out}}{2} \tag{4.80e}$$

The friction pressure drop Δp_f in equation (4.80b) is computed in the pressure drop model. An additional equation is formulated in the gas cell to compute the heat flow rate

$$\dot{Q} = (\alpha A)_{\mathit{eff}} (T_w - T) \tag{4.81}$$

where T_w is the wall temperature, T the gas temperature, and $(\alpha A)_{\mathit{eff}}$ the effective coefficient of heat transfer which is computed from

$$(\alpha A)_{\mathit{eff}} = |\dot{m}| c_p \exp\left(\frac{\alpha A}{|\dot{m}| c_p} - 1\right) \tag{4.82}$$

The derivation of this equation as presented by Tegethoff (1999) uses an analytical approximation of the gas temperature in the control volume based on the ideal gas law. The coefficient of heat transfer α in equation (4.82) is computed in the heat transfer model.

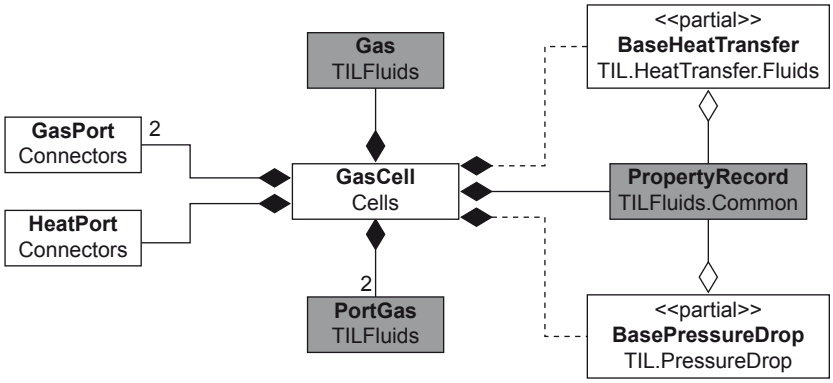


Figure 4.14: UML class diagram of `GasCell`. The two shown partial base models for heat transfer and pressure drop correlations are replaced when instantiating the refrigerant cell by one of the models presented in section 4.5.

The `PropertyRecord` from `TILFluids` and aggregation are used to propagate the fluid properties from the gas cell to the heat transfer and pressure drop models. The additional variables listed in table 4.2 are also propagated using aggregation. Two gas ports and a heat port define the interfaces of the gas cell. The most advanced heat transfer and pressure drop models for the gas cell use the correlations presented in section 4.5.3.

4.7.3 Liquid Cell

The new component model library also provides liquid cells to model incompressible fluid flow. Figure 4.15 shows the class diagram of a liquid cell. A liquid model `Liquid` from `TILFluids` is used to model the fluid properties in the control volume. The liquid cell uses the same set of equation as the gas cell presented in the previous section except for a transient formulation of the energy balance.

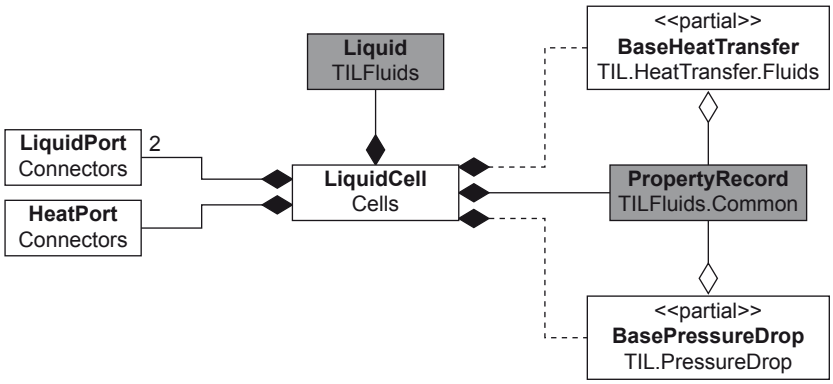


Figure 4.15: UML class diagram of `LiquidCell`.

4.7.4 Wall Cell

The wall cell is used to model walls separating fluid flows in tubes and heat exchangers. Figure 4.16 shows the class diagram of a wall cell. A replaceable material model is used to compute all material properties. A `PropertyRecord` is used to propagate the material properties to the heat transfer model. Four heat ports define the interfaces to the neighboring wall and fluid cells.

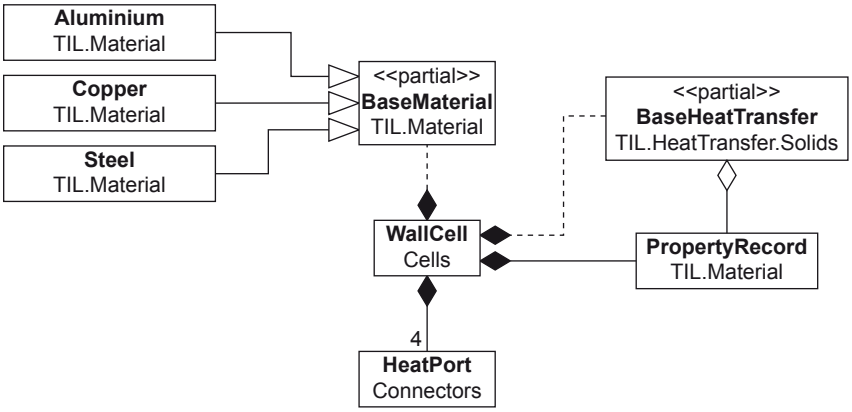


Figure 4.16: UML class diagram of `WallCell`.

The four heat ports in the wall cell are labeled using the cardinal points (e.g., `heatPortN` for the heat port at the north side). Two different thermal resistances R_{NS} and R_{WE} can be specified within the wall cell representing the thermal vertical and horizontal thermal resistance of the cell. The four heat flows are computed from

$$\dot{Q}_{N,S} = 2\,\frac{T_{N,S} - T}{R_{NS}} \tag{4.83a}$$

$$\dot{Q}_{W,E} = 2\,\frac{T_{W,E} - T}{R_{WE}} \tag{4.83b}$$

where T is the temperature of the wall material in the center of the wall cell. The energy balance is formulated as

$$mc_v\frac{dT}{dt} = \dot{Q}_N + \dot{Q}_E + \dot{Q}_S + \dot{Q}_W \tag{4.84}$$

where c_v is the specific heat capacity of the wall material and m the mass of the cell.

4.8 Heat Exchangers

The heat exchanger model is the most complex model in the component model library and was designed based on the experiences of earlier projects by Kossel (2005) and Ahlbrink (2007). The major design goal was to create a heat exchanger library that is both, simple in its structure and easy to extend and customize. This section explains the basic concept and the structure of the heat exchanger models.

Many different approaches are possible when modeling heat exchangers in Modelica. Schiavo and Casella use finite-element models derived using the Galerkin/Least-Squares approach

to describe homogeneous two-phase flows in heat exchangers (Schiavo and Casella, 2007). The heat exchanger model included in `Modelica_Fluid` consists of two discretized pipe models connected to a wall element (Casella et al., 2006). The pipe models use a spatial discretization according to the finite-volume method (Elmqvist et al., 2003). There are two different pipe models in `Modelica_Fluid`, `DistributedPipe` including n volumes with dynamic mass and energy balances and static momentum balances on a staggered grid and `DistributedPipeLumpedPressure` with n dynamic energy balances but only one mass balance (one pressure) and two momentum balances (two flow rates). Combining the pressure states in a distributed pipe into one single state is one possibility to avoid stiff models but only works when the distributed friction is small. The approach discussed in section 4.3 is another possibility to avoid a large number of fast pressure states. Using this approach of locally constant time derivatives of pressure yields to control volumes where the specific enthalpy h is the only differential variable.

This section uses the model of a fin-and-tube heat exchanger to explain the basic structure and the design of the heat exchanger model. All other heat exchangers are composed in a similar way.

4.8.1 Sandwich Concept

All heat exchanger models in the new component model library are composed using cells to model the two fluid flows and the separating wall as shown in figure 4.17 for the fin-and-tube heat exchanger. Three cells can be combined into a single basic element that is used to discretize the heat exchanger. The resulting structure is called *sandwich structure* by Tegethoff (1999) and is described in detail by Kossel (2005) in a Modelica context. It offers both, a simple structure and high flexibility.

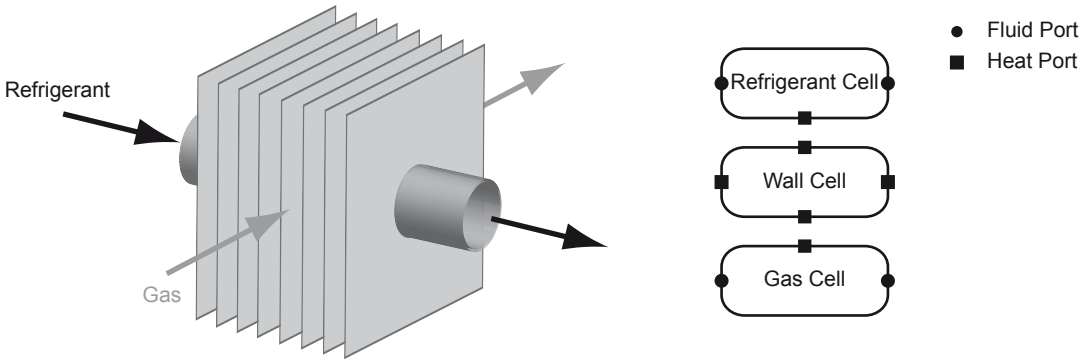


Figure 4.17: Basic element for the `FinAndTube` heat exchanger in the new component model library.

A basic element contains a refrigerant cell, a wall cell, and a third cell depending on the type of the heat exchanger. The different types of heat exchangers in the `HeatExchangers` package are listed in table 4.5. The cells are explained in section 4.7. Each heat exchanger instantiates n primary, n wall, and n secondary cells and connects them depending on the flow pattern that can be counter flow, co-current flow, or cross flow.

Heat Exchanger	Primary Cell	Wall Cell	Secondary Cell
FinAndTube	RefrigerantCell	WallCell	GasCell
FinAndTube_moistAir	RefrigerantCell	WallCell	MoistAirCell
IHX	RefrigerantCell	WallCell	RefrigerantCell
TubeInTube_liquidRef	RefrigerantCell	WallCell	LiquidCell

Table 4.5: Structure of a basic element for different types of heat exchangers in the new component model library.

Figure 4.18 shows the structure of the heat exchanger models for all flow patterns. The cell numbering is designed in the most natural way in which only cells with equal numbers are connected vertically. Different models exists for co-current and counter flow heat exchangers though this is not mandatory due to the application of `semiLinear()` in each fluid cells (see section 4.2). One advantage of having two separate models instead of a single model is that the two separate models can have unique icons in the icon layer making it much easier for users to distinguish between the two models. Another advantage is that the cells can be connected differently in the two models to ensure that the fluid port called `inlet` in each cell model is the inlet port for the designed flow direction.

The cross flow heat exchanger model uses a splitter to split the entering gas mass flow rate into n equal mass flow rates and a joiner to unite the n gas mass flow rates again. It is also possible to use a distribution matrix to split the entering gas mass flow rate to model a non-uniform incident air flow. This was successfully tested by Mazen (2007) using models from the new component model library.

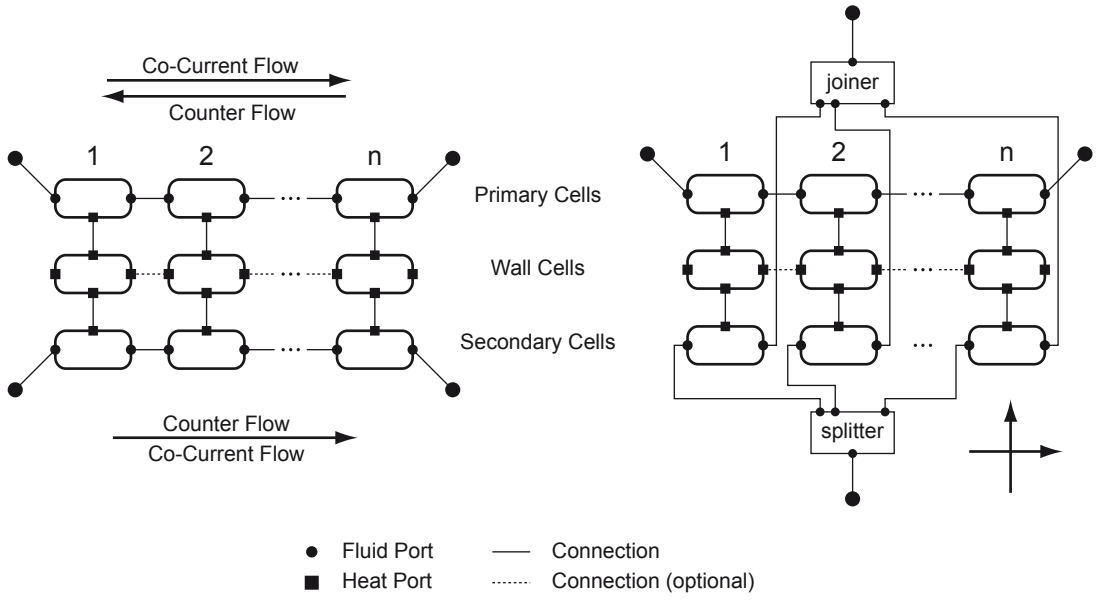


Figure 4.18: Structure of heat exchanger models. The left-hand side shows the structure for co-current and counter flow heat exchangers, the right-hand side for cross flow heat exchangers.

4.8.2 Class Structure

Figure 4.19 shows the class diagram of the fin-and-tube heat exchanger model. All cells are instantiated in the `BaseHX` model. The connections are made in the derived classes depending on the flow pattern (e.g., counter flow). The light gray classes represent the inheritance tree which is only two levels deep. The heat exchanger model extends from `PartialComponent` that is explained in detail in section 4.10.3. The dark gray classes are medium models from `TILFluids` that are used to compute additional fluid properties at the inlets and outlets of the heat exchanger.

All heat transfer and pressure drop models are defined as replaceable local model in the heat exchanger model. The modifiers of the cells are used to propagate the selected models to the cells. The same technique is used for the wall and for the fin material model. The geometry record is used to store all geometrical information and to compute the discretized quantities. The discretized quantities (e.g., the cell volumes) are propagated to the cell models using the modifiers of the cells. Aggregation is used to make the geometry record available in each heat transfer and pressure drop model by using the `inner`/`outer`-concept as explained in section 2.3.2.

A summary record is used to collect the most important information on the heat exchanger making it easier for the user to analyze the numerical results. Note that more complex heat exchangers can also be modeled using the presented structure by either combining several of the available heat exchanger models into a new more complex heat exchanger model or by connecting the cells manually representing the structure of the complex heat exchanger to be modeled.

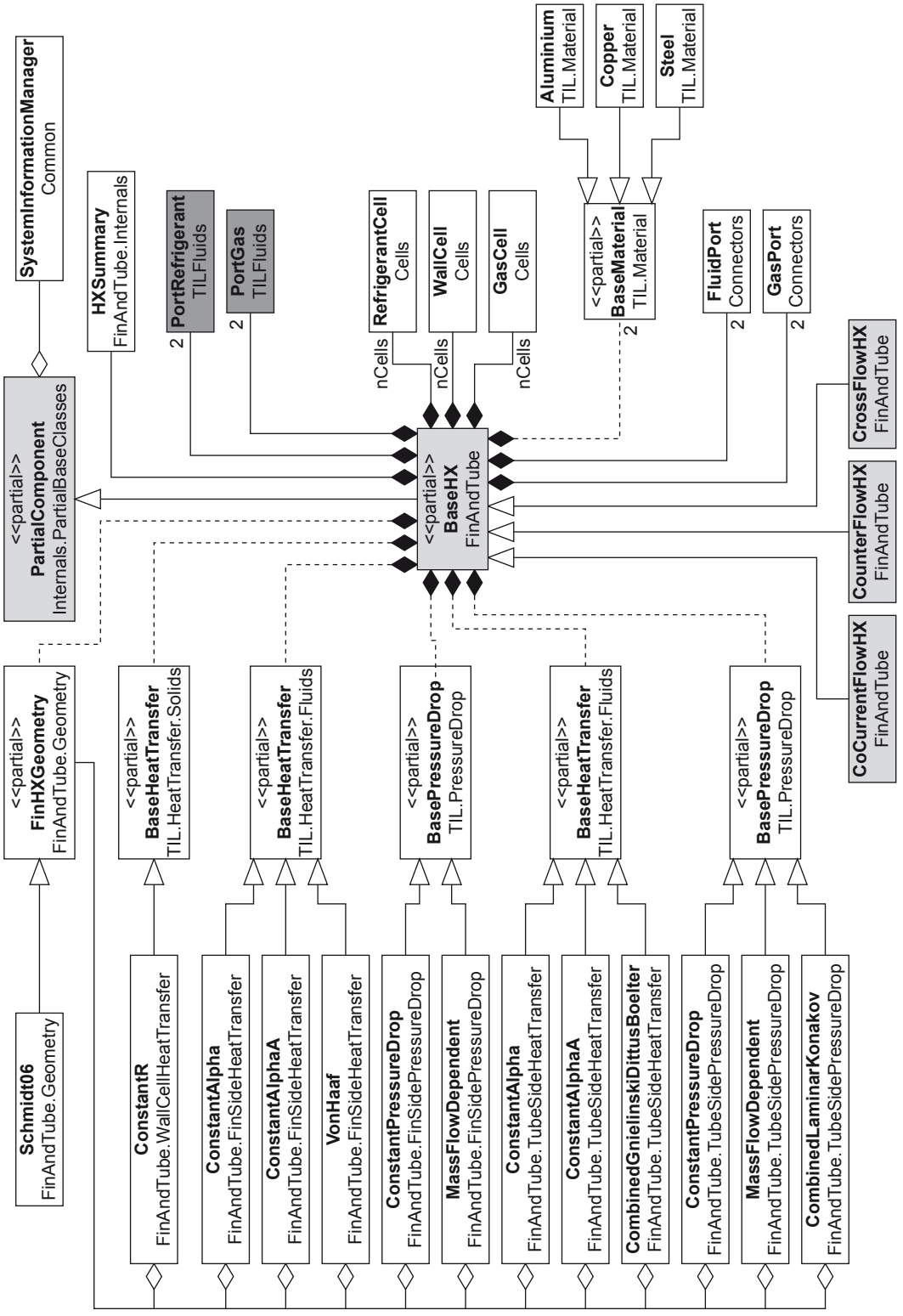


Figure 4.19: UML class diagram of `FinAndTube` heat exchanger in the new component model library. Class diagrams of the cells can be found in figures 4.13, 4.14, and 4.16.

4.8.3 Initialization

The initialization of all heat exchanger models is handled in the base class of the specific heat exchanger (e.g., `FinAndTube.BaseHX`) using a string parameter called `initialization`. Table 4.6 shows the implemented initialization methods for heat exchangers.

Parameter Value	Description
`"none"`	No initial equation is formulated.
`"steady-state"`	The heat exchanger is initialized in steady-state, i.e., $dh/dt = 0$ in each refrigerant cell.
`"constant enthalpy"`	The heat exchanger is initialized with constant specific enthalpy, i.e., $h = const$ in each refrigerant cell. The user has to specify a value for the specific enthalpy.
`"linear constant enthalpy"`	The heat exchanger is initialized with a linear profile for the specific enthalpy in the refrigerant cells. The user has to specify the specific enthalpy to be used for the first and the last refrigerant cell.
`"constant temperature"`	The heat exchanger is initialized at constant temperature, i.e., $T = const$ in each refrigerant cell. The user has to specify a value for the temperature.

Table 4.6: Initialization methods for heat exchanger models.

4.9 Compressor Model

Compressor models with different levels of detail are available in the `Compressors` package. One of the advanced models describes a swash plate compressor based on characteristic curves for the compressor efficiencies. This model was initially developed by Försterling (2003). An improved model was first published by Tummescheit et al. (2005a) though it was also developed by Försterling as stated in Tummescheit et al. (2005b). The model uses three evaluation parameters to describe the compressor regarding the mass flow rate $\dot{m}_{\mathit{eff}}$, the driving power P_{eff}, and the discharge temperature T_d: the volumetric efficiency λ_{eff}, the effective isentropic efficiency $\eta_{\mathit{eff},is}$, and the isentropic efficiency η_{is} defined as

$$\lambda_{\mathit{eff}} = \frac{\dot{m}_{\mathit{eff}}}{V n \varrho(p_s, T_s)} \tag{4.85}$$

$$\eta_{\mathit{eff},is} = \frac{P_{is}}{P_{\mathit{eff}}} = \frac{(h_{d,is} - h_s)\dot{m}_{\mathit{eff}}}{2\pi |M| n} \tag{4.86}$$

$$\eta_{is} = \frac{h_{d,is} - h_s}{h_d - h_s} \tag{4.87}$$

where V is the displacement volume, n the speed, and M the torque. The subscript d refers to the discharge side, s to the suction side, is to isentropic conditions, and eff to effective values. The efficiency functions are factored in two parts. One part $f(\pi, n)$ captures the influence of the pressure ratio and rotational speed and a second part $g(x, n)$ to take into account the

control of the swash plate angle and rotational speed. Due to this separation, a derivation of efficiencies for the full load case is possible even if measurements for the influence of the swash plate are not available. The following relation is used for all efficiencies

$$\lambda, \eta(\pi, n, x) = f(\pi, n) \cdot g(x, n) \quad \text{with} \quad g(\pi, x = 1) = 1 \tag{4.88}$$

The following correlations are used to describe the full load case (i.e., $g(\pi, x = 1) = 1$) for a CO_2 swash plate compressor

$$\lambda_{\mathit{eff}}(\pi, n) = \left(\frac{\pi_0 - \pi}{\pi_0 - 1}\right)^2 (a_2 n^2 + a_1 n + a_0) \tag{4.89}$$

with three coefficients and one constant. The same correlation can be used to describe the effective isentropic efficiency $\eta_{\mathit{eff},is}$ and the isentropic efficiency η_{eff}

$$\eta(\pi, n) = b\ \frac{\pi_0 - \pi}{\pi_0} - b\ c\ \left(\frac{1}{c}\ \frac{\pi_0 - 1}{\pi_0}\right)^{\pi} \tag{4.90}$$

where

$$b(n) = b_1 n + b_0 \quad \text{and} \quad c(n) = c_1 n^{c_2} \tag{4.91}$$

with four coefficients and one constant. The coefficients for equation (4.89) and (4.90) have to be adapted to measurement data. Fitted sets of coefficients for various compressors are not included in the `Compressors` package but are provided in an add-on library.

4.10 Basic Component Models

This section describes additional component models. The first described component model is the system information manager that handles information about the entire system to be modeled. The second model is the pressure state component model which is used to compute the time derivative of pressure required for the specific simplification of the balance equations described in section 4.3. The third described component model is the partial base component model that all other component models extend from. The last component models described in this section are the boundary models used for all sources and sinks in testers for components as well as in system simulations.

4.10.1 System Information Manager

Each system requires basic information including technical information (e.g., the used refrigerants, gases, and liquids) as well as numerical information (e.g., the total refrigerant mass) that are required for a simulation. The new component model library uses a component called `sim` of type `SystemInformationManager` to store all required information in one central place for each system. It is furthermore used to compute the total refrigerant mass and the total inner volume which are important system properties. Aggregation is used to propagate information from the system information manager to the system and vice versa as described in section 2.3.2. The `SystemInformationManager` contains one `SIMPort` (see code listing 4.4) that is used to determine the total refrigerant mass and volume and to propagate

the number of refrigerants used in the system. Each component model has access to the system information manager via the definition of an outer `SystemInformationManager` named `sim` (see also section 4.10.3). Besides the definition of media required for the simulation, the system information manager also allows for the specification of post-processing parameters (e.g., color schemes).

```
import SI = Modelica.SIunits;

connector SIMPort "System information manager port"
  parameter Integer nRefrigerants "Number of refrigerants used in the system";

  flow SI.Mass[nRefrigerants] refrigerantMass "Accumulated refrigerant mass";
  flow SI.Volume[nRefrigerants] refrigerantVolume "Accumulated refrigerant volume";
end SIMPort;
```

Code Listing 4.4: Code for `SIMPort`.

Figure 4.20 shows how multiple instances of `SystemInformationManager` can be used in a single simulation. Air-conditioning systems are independently instantiated in each railway car. Each air-conditioning system has its own system information manager named `sim`. This is possible due to the `inner`/`outer`-concept in Modelica where an outer element always references the inner element with the same name that is nearest in the enclosing instance hierarchy of the outer element declaration. It would therefore also be possible to instantiate the entire train in a model of a railway station where the railway station itself contains a system information manager for the air-conditioning system of the building itself.

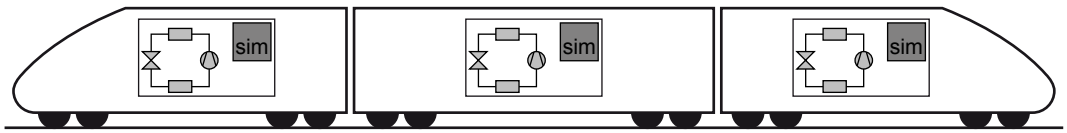


Figure 4.20: Multiple instances of `SystemInformationManager` in a single simulation.

4.10.2 Pressure State

The time derivative of pressure is assumed to be constant in flow direction at each pressure level as explained in section 4.3. The derivative is handled in a model called `PressureState` which is sketched in code listing 4.5. Different concepts have been tested to propagate the computed time derivative from the `PressureState` component to other components as discussed in the following paragraphs.

```
model PressureState "Component to determine time derivative of pressure"
  Connectors.FluidPort inlet "Inlet port";
  Connectors.FluidPort outlet "Outlet port";

  Real dpdt "Time derivative of pressure";
equation
  dpdt = der(inlet.p); // the time derivative of pressure is handled by this equation;
```

```
                    // inlet.p is a state variable

  inlet.m_flow + outlet.m_flow = 0 "Static mass balance";
  inlet.H_flow + outlet.H_flow = 0 "Static energy balance";
  inlet.p = outlet.p "Static momentum balance";

  ... // further code omitted
end PressureState;
```

Code Listing 4.5: Modelica code for `PressureState` from `HVAC_p` package.

One straightforward concept to propagate the *dp/dt*-information is to use the fluid ports. The main disadvantage of this concept is that each component has to process the provided information even if this information is of no importance for the specific component (e.g., in simple valve and compressor models) making it harder to understand the system of equations for developers as well as for users. A second concept is the introduction of an additional connector to propagate the derivative information. This concept would allow the developer to decide for each component whether this additional connector is required or not. The main disadvantage of this concept is that the additional connectors have to be manually connected in each system which is cumbersome and quite confusing for new users. The third concept is to use the `inner/outer`-concept in Modelica to propagate the derivative information among components. This concept neither requires additional explicit connections nor each model to consider the information. The *dp/dt*-approach explained in section 4.3 requires that a pressure state is computed at each pressure level in a system. The system information manager explained in section 4.10.1 is used to store the time derivatives of the different system pressures. The numbers (1) and (2) in figure 4.1 refer to the two independent pressure states in the CO_2 refrigeration system computed in the two `PressureState` components.

4.10.3 Partial Base Component

All components in the `HVAC_p` package extend from the partial base component model `Partial-Component`. Unlike in other component libraries for the computation of thermodynamic systems (e.g., Modelica_Fluid, ACLibrary) no conservation laws are formulated in this base model. Instead, all conservation laws have to be formulated in the component model itself. This allows for a component-specific formulation of the conservation laws and makes it as simple as possible for new users and developers to understand the component models. The `PartialComponent` model provides standardized access to the system information manager (see section 4.10.1) and additional information for post-processing or external GUIs. Code listing 4.6 shows the Modelica code for the `PartialComponent`.

```
import TIL.HVAC_p.Common.*;

partial model BaseComponent "Partial base model for all components"
  parameter ComponentType componentType "Component type";
  parameter RefrigerantID refrigerantID "Refrigerant ID";

  ... // further code omitted
```

```
protected
  outer SystemInformationManager sim "System information manager";
  SIMPort simPort(
   final nRefrigerants=sim.nRefrigerants) "SIM port";
equation
  connect(sim.simPort,simPort);
end BaseComponent;
```

Code Listing 4.6: Code for `PartialComponent`.

`ComponentType` is an enumeration that uniquely identifies the type of the component model (e.g., `Boundary`, `OrificeValve`) allowing external GUIs and post-processing tools like the StateViewer (see section 5.3) to determine the type of an instance from the result file. The `BaseComponent` also provides access to the system information manager and connects the local `SIMPort` to the `SIMPort` of the system information manager using implicit connections briefly discussed in section 2.6.1. The `refrigerantID` specifies which refrigerant defined in the system information manager is used in the component. If more than one refrigerant is used, an additional parameter such as `refrigerantID2` has to be defined. Similar parameters are used in components utilizing gases or liquids.

An additional partial base model for components `PartialVolumelessComponent` extends from `PartialComponent` and sets the volume and the refrigerant mass of the component to zero.

4.10.4 Boundaries

All sink and source elements for fluids are called boundaries in the new component model library. Boundary component models are for example used in the CO_2 refrigeration system presented in figure 4.1 to specify the air-side boundary conditions for each of the three heat exchangers. Different boundary models exist for refrigerants, gases, and liquids using the corresponding ports (see section 4.2) and fluid models from `TILFluids` (see table 3.3). The type of the boundary is specified by a string parameter called `boundaryType` defined in each boundary model. All possible values for `boundaryType` are listed in table 4.7.

Identifier	Description
"free"	No parameter is fixed at the boundary. This boundary condition represents a reservoir at fixed temperature and unspecified pressure.
"p"	The pressure is fixed at the boundary. This boundary condition represents a reservoir at a fixed temperature that impresses a fixed pressure on the system.
"m_dot"	The mass flow rate is fixed at the boundary. This boundary condition represents a reservoir at a fixed temperature with a fixed mass flow rate. The boundary can be used as a source (negative mass flow rate) or sink (positive mass flow rate) element.
"p, m_dot"	All parameters are fixed at the boundary. This boundary condition represents a reservoir at a fixed temperature and pressure with a fixed mass flow rate. The boundary can be used as a source (negative mass flow rate) or sink (positive mass flow rate) element.

Table 4.7: Values for `boundaryType` to specify the type of a fluid boundary element. Note that the specific enthalpy h or the temperature T always have to be specified to allow for flow reversal.

Each boundary defines two medium models: one at the port of the boundary and the other one representing the properties at the outside. The medium model at the port uses the pressure and the specific enthalpy in the connector instance to compute all other medium properties of the fluid leaving or entering the boundary. The second medium model uses the port pressure and a specified temperature or specific enthalpy to compute the fluid properties at the outside. The two fluid models in each boundary are identical if the mass flow rate in the port is negative (i.e., the boundary is a source).

Additional boundary elements that can be connected to heat ports are also provided in the `Boundaries` package. Table 4.8 lists the different types for these boundaries.

Identifier	Description
"free"	No parameter is fixed at the boundary. This boundary condition represents a reservoir at unspecified temperature.
"T"	The temperature is fixed at the boundary. This boundary condition represents a reservoir at a fixed temperature.
"Q_dot"	The heat flow rate is fixed at the boundary. This boundary condition represents a reservoir with a fixed heat flow rate. The boundary can be used as a source (negative heat flow rate) or sink (positive heat flow rate) element.
"T, Q_dot"	All parameters are fixed at the boundary. This boundary condition represents a reservoir at a fixed temperature with a fixed heat flow rate. The boundary can be used as a source (negative heat flow rate) or sink (positive heat flow rate) element.

Table 4.8: Values for `boundaryType` to specify the type of a heat boundary element.

4.11 Numerical Aspects

The demonstrating example presented in the introduction to this chapter is used to discuss some of the benefits of the handling of numerical states in the new component model librray. The CO_2 refrigeration system sketched in figure 4.1 consists of ten components on the refrigerant side. Each heat exchanger is modeled using ten refrigerant cells, ten gas cells, and ten wall cells in a sandwich structure as described in section 4.8. The numerical results obtained from the simulation are shown in a pressure-enthalpy diagram and in a temperature-entropy diagram in figures 5.3 and 5.4 respectively. The resulting system of equations describing the real-world problem is a DAE (differential algebraic equation) that can be written in its implicit form as

$$F(x(t), \dot{x}(t), y(t), t) = 0 \tag{4.92}$$

where t is the time, $x(t)$ are the state variables, and $y(t)$ the algebraic variables. The DAE in equation (4.92) is called linear-implicit if it is linear in its derivatives $\dot{x}(t)$ in which case it can be written as

$$A(x(t), y(t), t)\ \dot{x}(t) + f(x(t), y(t), t) = 0 \tag{4.93}$$

This DAE can be transformed in a semi-explicit non-linear form if $A(x(t), y(t), t)$ in equation (4.93) is not singular yielding

$$\dot{x}(t) = f(x(t), y(t), t) \tag{4.94a}$$

$$0 = g(x(t), y(t), t) \tag{4.94b}$$

The main difference between a DAE and an ODE (ordinary differential equation) are the algebraic constraints $g(x(t), y(t), t)$ that very often restrict the number of degrees of freedom, i.e., not all differentiated variables are numerical states, resulting in a higher order problem. Index reduction is applied to transform the higher order system into an index one problem that can be solved numerically. Much more detailed descriptions of this numerical procedure can be found in the standard literature such as Brenan et al. (1996) or Hairer and Wanner (1996). The demonstrating example from figure 4.1 does not require index reduction due to the specific formulations of the conservation laws discussed in section 4.3, thus simplifying the process of numerical transformations for the simulation environment.

From a numerical point of view, many semi-explicit DAEs can be solved like an ODE if the initial conditions are known. The initial conditions x_0 and y_0 that are sometimes hard to find (see section 5.4) have to be consistent with the algebraic constraints $0 = g(x_0, y_0, t_0)$. A basic requirement is that the Jacobian

$$\frac{\partial g(x, y)}{\partial y} \tag{4.95}$$

is invertible near the solution of equation (4.94a).

The DAE system resulting from the CO_2 refrigeration system sketched in figure 4.1 contains 63 numerical state variables distributed among the components as listed in table 4.9.

The major difference between the new component model library and other Modelica libraries is the number of pressure states. The pressure states are often the fastest states in thermo-fluid systems and are strongly coupled to the mass flow rates (see section 4.5) and can cause a stiff DAE system, i.e., a system where the eigenvalues of the Jacobian $\partial f/\partial x$ differ in magnitude by a factor of 10^4 to 10^6. The *dp/dt*-approach that is described in section 4.3 reduces the number of pressure states to a single numerical state per pressure level. It is important to notice that this simplification does not yield a constant pressure at each pressure level. The pressure distribution in the heat exchanger model for example still depends on the selected pressure drop model and thus on the mass flow rate. A detailed analysis of the differences between the numerical approach used in the new component model library and the approaches used in other component model libraries is a very interesting topic but goes beyond the scope of this work.

Component Name	**No. of States**	**States**
`compressor`	-	
`pressureState1`	1	Pressure at inlet of pressure state component 1
`gasCooler1`	10	Specific enthalpy in each refrigerant cell
	10	Temperature in each wall cell
`tube`	-	
`gasCooler2`	10	Specific enthalpy in each refrigerant cell
	10	Temperature in each wall cell
`valve`	-	
`evaporator`	10	Specific enthalpy in each refrigerant cell
	10	Temperature in each wall cell
`accumulator`	1	Specific enthalpy of refrigerant
`pressureState2`	1	Pressure at inlet of pressure state component 2
`sim`	-	
Total	**63**	

Table 4.9: States in CO_2 refrigeration system from figure 4.1.

Chapter 5

Visualization

Visualization is an important part of the process of model development, model verification and validation, and simulation. This chapter presents a tool for the automated generation of class diagrams and explains the unique features of the object-based fluid property modeling approach presented in chapter 3 that allows for an automated generation of thermodynamic plots during and after the simulation.

5.1 Introduction

The visual abilities belong to the strongest cognitive abilities of human beings. Visualization of complex structures or numerical results is of great importance when working with simulations and allows for a fast and reliable detection and interpretation of sources of error. As pointed out in chapter 2, a profound object-oriented analysis before and during the development process of an object-oriented model library supports the developer in developing a library with a simple yet powerful object-oriented structure. The same analysis helps new developers when starting to maintain or extend an existing library. It can also support users to better understand the library they are working with. Generating class diagrams as visual representation of the hierarchical structure is thus a very important task during the entire life cycle of an object-oriented model library. Nevertheless, no tool currently exists to generate those diagrams automatically from Modelica code. Section 5.2 presents a simple tool that was developed to demonstrate the benefits of an automated generation of class diagrams. The tool features a proprietary Modelica parser and should be seen as a proof-of-concept.

Of even greater importance is the visualization of numerical results during the development process of component models as well as during and after the simulation of entire systems. The 3-dimensional visualization of numerical results from mechanical models in Dymola using imported CAD meshes is a good example for a visualization that is strongly coupled with the simulation. Engineers working a lot with thermodynamic systems are used to analyze components and systems in phase diagrams (e.g., pressure-enthalpy diagrams, temperature-entropy diagrams). There are no software tools available that support the automatic generation of phase diagrams from numerical results. Instead, existing libraries to simulate thermodynamic

systems such as the `AirConditioning` library by Modelon add the thermodynamic visualization of results as an afterthought that is quite limited in its functionality. `TILFluids` was especially designed for an easy support of thermodynamic visualizations that can be set-up automatically and that can be used in the post-processing process but also during the simulation and even during initialization. This approach is described in section 5.3.

5.2 Automated Generation of UML Class Diagrams

The object-oriented analysis is a very important tool before and during the development and extension of object-oriented component libraries as pointed out in chapter 2. The result of this analysis is a graphical representation of the hierarchical structure of a Modelica library. Many different formats exist for this graphical representation. The UML standard that is introduced for Modelica in chapter 2 is a widely used standard. Many specific tools exist for the automated generation of UML class diagrams for various object-oriented programming languages but not for Modelica. ModelicaCDV (Modelica Class Diagram Visualizer) is a first attempt to develop a class diagram visualizer for Modelica libraries and is described in detail by Loeffler (2006) (see also Loeffler et al., 2006). ModelicaCDV uses its own Modelica parser to transform the Modelica source code into an internal data representation. ModelicaCDV is written in Java and uses the GUI toolkit Swing (Sun Microsystems, Inc.). Figure 5.1 shows a screenshot of ModelicaCDV visualizing the example for an exchangeable base class as shown in figure 2.9.

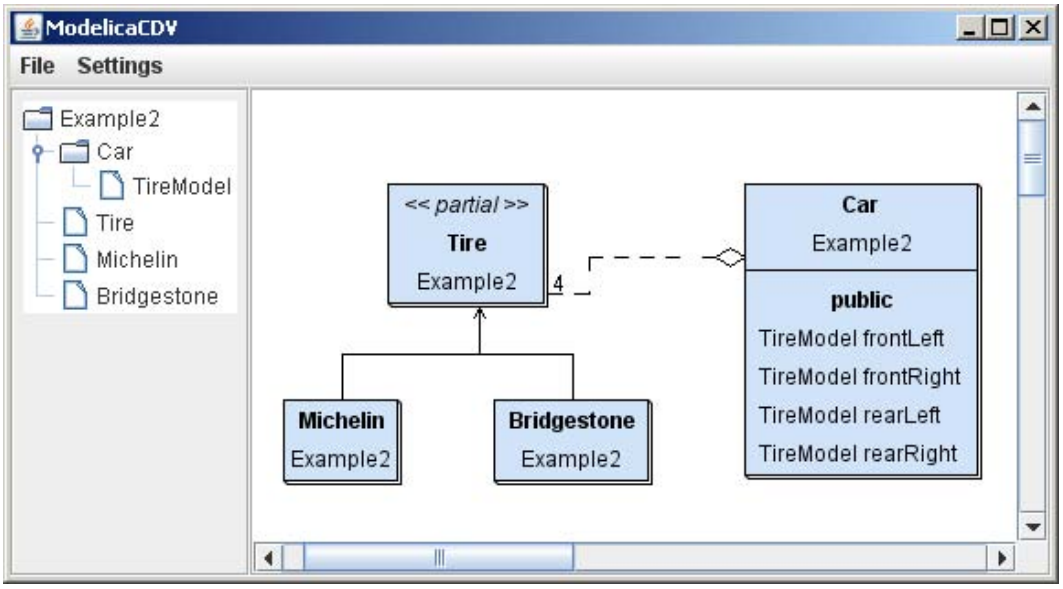


Figure 5.1: Screenshot of ModelicaCDV showing example from figure 2.9.

The UML specifies the elements in class diagrams and their graphical representation but does not prescribe how classes and associations should be arranged. There exist a number of common criteria for the generation of *nice* graphs: (compare Siebenhaller, 2003)

- *Edge Length*: Edges should be as short as possible. The longer an edge, the harder it is to conceive the connection.
- *Area*: The area required to display a graph should be minimized to allow understanding its structure at a glance. The smaller the graph, the closer positioned the classes are and the shorter the edges.
- *Intersecting Edges*: Intersecting edges make it harder to receive their path. The number of intersecting edges should be as small as possible. The ideal graph is planar.
- *Distance between Nodes*: The distance between nodes should be as small as possible. This also minimizes the edge lengths.

Some additional criteria for class diagrams significantly improve their readability and comprehensibility:

- *Hierarchy*: It is common practice in inheritance to place the derived class below the base class. This yields a tree structure and makes it easier to understand the hierarchy.
- *Labeling*: The text contained in class diagrams should always be horizontally aligned. This also applies to the cardinality of aggregations.
- *Edges and Nodes*: Edges represent inheritance and aggregation relationships. Classes that are related with each other should be arranged as close as possible so that it is possible to visualize the relationship without the need of scrolling. The size with which classes are displayed is always a trade-off between readability of the class name and attributes and the number of elements visible on a single screen.
- *Computational Efficiency*: The algorithms for the automatic layout of UML diagrams should be capable of shortly delivering a class diagram for graphs with a large number of nodes and edges.

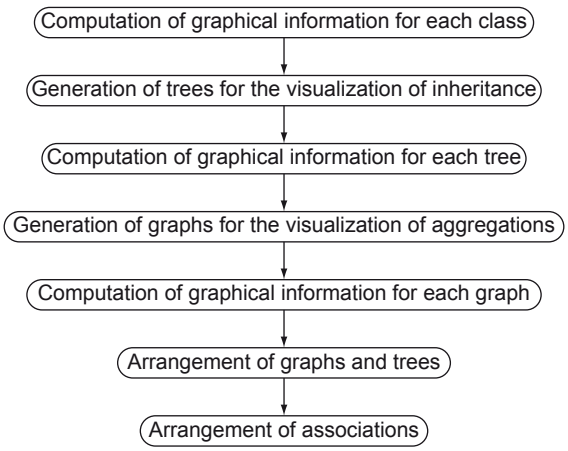


Figure 5.2: Layout algorithm used in ModelicaCDV.

Figure 5.2 shows the sequence of operations performed by ModelicaCDV when generating a class diagram. The first step after parsing is the computation of the layout for each class.

This step includes computing the size of the rectangle representing the class and the layout of the attributes as shown in figure 2.1. The next step only takes into account classes with inheritance relations and generates the trees representing the hierarchical structure of the inheritance relations. Classes that do not have inheritance relations are neither base classes nor derived classes. These classes can only be related to other classes via aggregation. In the next step, graphs for aggregations are computed where each edge represents an aggregation. All generated graphs (including the trees) are arranged separately and then combined in one class diagram in a final step.

The presented software for the automated generation of class diagrams could furthermore be extended to automatically check some of the design rules for object-oriented model libraries. The software could analyze the library based on the design rules 1, 4, and 6 and could generate a list of warnings whenever the design rules are violated. As a proof of concept, ModelicaCDV was extended to generate a list of warnings for all inheritance relations that are based on the Cardelli type system without an explicit `extent` statement as presented in section 2.3.3.

5.3 Thermodynamic Phase Diagrams

The visualization of numerical results in thermodynamic phase diagrams supports the developer when testing new models and helps the user to better understand simulated component models or entire thermodynamic systems. The advanced object-based approach used in `TILFluids` to model fluid properties which is described in section 3.5 supports the automated generation of thermodynamic phase diagrams. This section explains the basic concept and presents a simple example demonstrating the advantages of the new concept.

Each medium model in `TILFluids` contains an integer called `stateViewerIndex` that is used for the automated generation of thermodynamic phase diagrams. This index has to be specified for each medium object in a component model. A tool can then collect all thermodynamic information for the generation of a phase diagram by searching for these indices in the result file and by displaying the fluid properties of the corresponding medium object. The tool can also generate a cycle from the visualized medium objects by connecting state points with consecutive indices. The `stateViewerIndex` is automatically assigned in component models using the additional variable `index` in the connector definition (see code listing 4.1). In order to allow for an automated numeration, a starting point has to be specified. This starting index can be set in the boundary models described in section 4.10.4 if single components are simulated or with a special component shown in figure 4.1 in front of the compressor if closed cycles are simulated. Code listing E.1 in appendix E lists the Modelica code for this special component called `StateViewerInterface`.

The requirements for the software tool used to generate the thermodynamic phase diagram from the simulation results can be summarized as follows:

- search the result file for all variables called `stateViewerIndex` and store the corresponding parent items in a list
- use the fluid properties provided in the result file for each item with a `stateViewerIndex` to draw state points in the thermodynamic phase diagram
- connect points with consecutive indices
- use the time information stored in the result file to visualize transient processes

Different software tools are available to plot thermodynamic phase diagrams. For refrigerants, REFPROP (Lemmon et al., 2007) or CoolPack (Andersen et al., 1999) can be used to generate various thermodynamic phase diagrams. The `AirConditioning` library provides specified classes that allow for the visualization of a pressure-enthalpy diagram in Dymola for a couple of refrigerants. Some other tools generate thermodynamic phase diagrams in Microsoft Excel (e.g., ThermoFluids by Wagner and Overhoff, 2006). None of the commercially available software tools is able to fulfill the requirements presented in the beginning of this paragraph. Therefore, a new software tool called StateViewer was developed to generate thermodynamic phase diagrams (see TLK-Thermo GmbH, 2007).

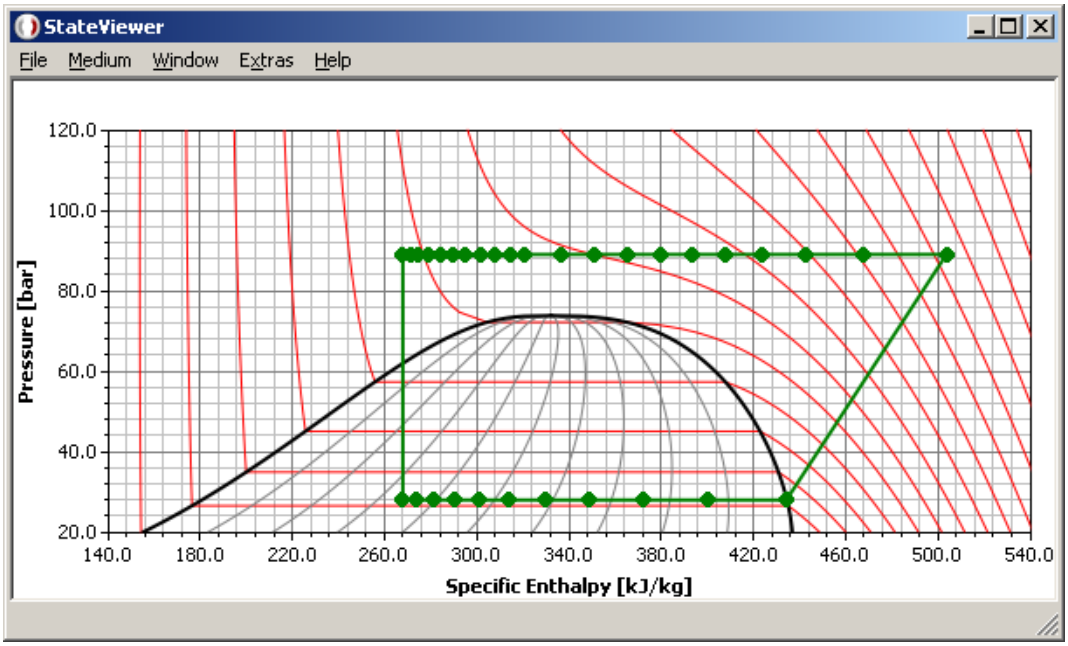


Figure 5.3: Pressure-enthalpy diagram of CO_2 refrigeration system from figure 4.1 in StateViewer.

Figure 5.3 shows a screenshot of the developed software tool visualizing the numerical results from the CO_2 refrigeration system shown in figure 4.1. The StateViewer uses the C++ interface to the TILFluidsLib presented in section 3.5.2 to generate the thermodynamic phase diagrams and can therefore display thermodynamic phase diagrams for all media available in the TILFluidsLib.

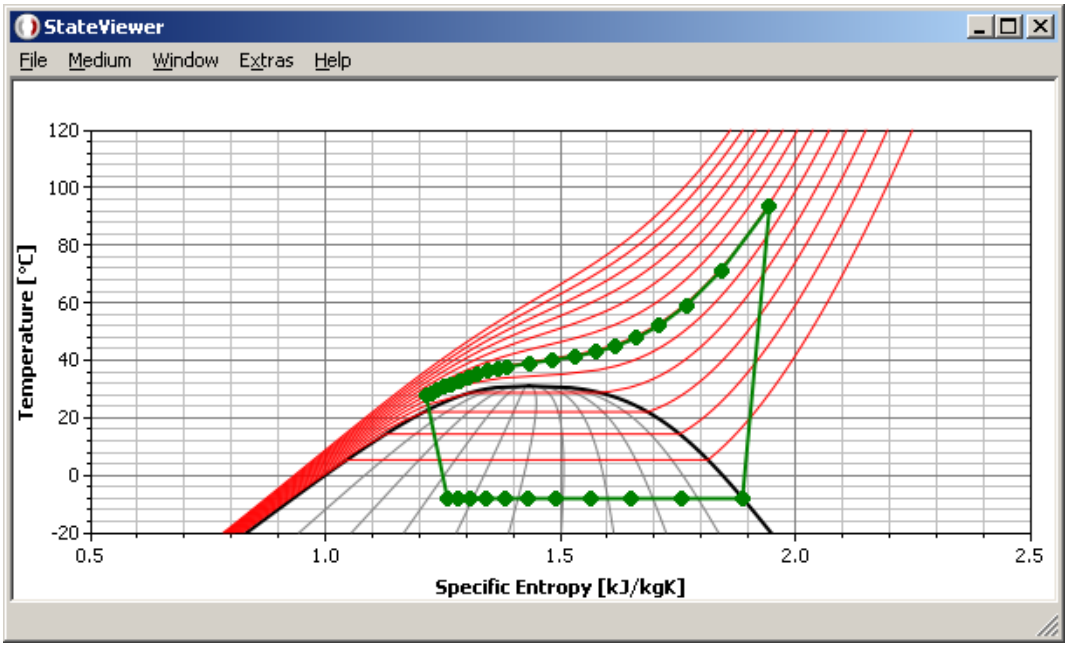


Figure 5.4: Temperature-entropy diagram of CO_2 refrigeration system from figure 4.1 in StateViewer.

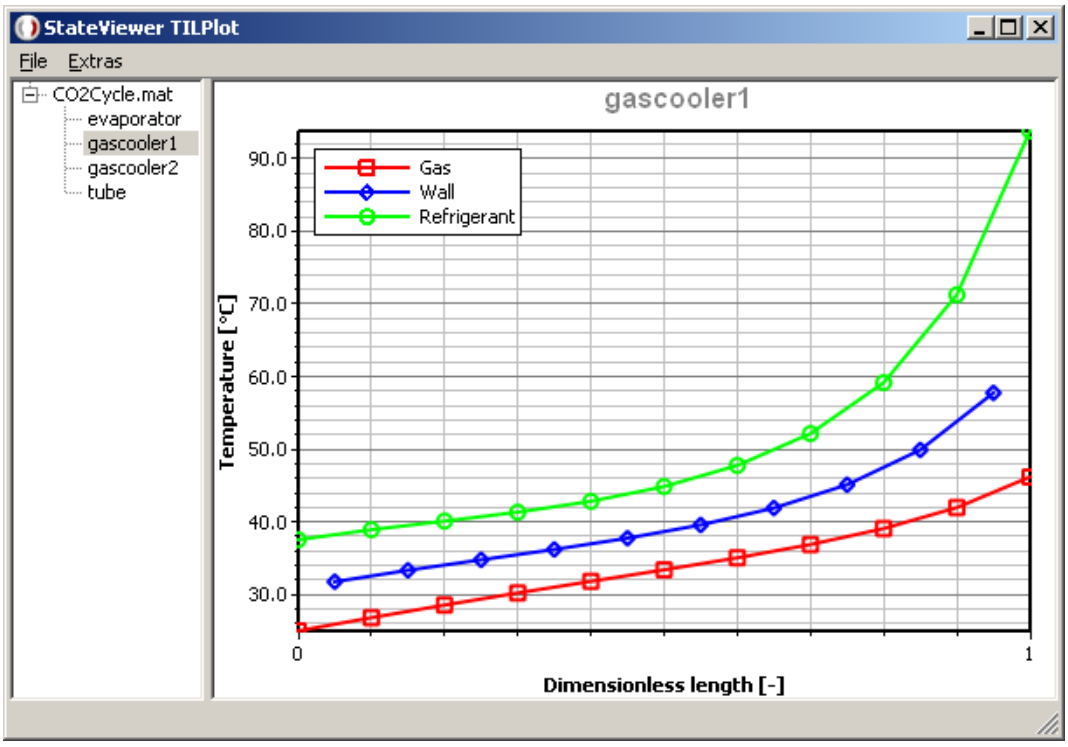


Figure 5.5: Temperature distribution in `gascooler1` of CO_2 refrigeration system from figure 4.1 in TILPlot.

The StateViewer offers many other convenient features that further simplify analyzing numerical simulation results. One example are plots of the temperature distribution in heat exchangers from the new component model library that can be provided in an additional

StateViewer window called TILPlot. Figure 5.5 shows the temperature distribution in the first gas cooler in the CO_2 refrigeration system shown in figure 4.1 that was automatically generated from the simulation results. The component type is determined from the corresponding parameter in the heat exchanger model as explained in section 4.10.3. Specialized plots like that are provided for many component models enabling the developer to check the model behavior and the user to better understand the numerical results.

5.4 Online Visualization during Initialization

Specifying good initial values to allow for the initialization of a simulation of a thermodynamic system can be a time-consuming procedure (see for example Raiser, 2005). One drawback of the Modelica language is the missing standardization of methods to influence the initialization process (e.g., by allowing for the explicit specification of tearing variables). A graphical representation of the numerical solution during initialization that is supported by the new component model library as well as by the fluid property library is a powerful tool to allow for quick error detection and improvements of initial values. The middleware TISC is used for the communication between the fluid property library `TILFluids` and the StateViewer. TISC is a co-simulation environment for exchanging data between different simulation applications. Kossel et al. (2006) give a good introduction to TISC. Some of the numerical benefits, mainly the advantages of a multi-rate integration, are presented by Tegethoff et al. (2007). `TILFluids` uses TISC to send all computed values including the time information to the StateViewer which is possible during transient simulations but also during initialization. Note that the indices for the StateViewer included in the fluid port definition (see code listing 4.1) and used for the automatic numbering of medium objects as explained in the beginning section 5.3 result in a system of equations that is solved during initialization. The connection of the visualized medium objects in the thermodynamic phase diagram has thus to be reevaluated for each sent value of a `stateViewerIndex` during initialization. The graphical representation of the simulated system during initialization helps the user to find medium objects that might cause numerical problems due to fluid property computations in invalid regions. The user can resolve this kind of problems by supplying reasonable start values to the affected medium objects.

A simple example illustrates the possibilities of the presented approach. The example uses component models from the `HVAC` package that is part of the new component model library. All models in this package are steady state models and are explained in appendix C. The `LoopBreaker` is required in the simulated system due to an algebraic loop for the mass flow rates which is characteristic for pure steady-state systems. The resulting triangular cycle is shown in the StateViewer screenshot in figure 5.6 in a pressure-enthalpy diagram. The resulting system of equation contains two non-linear equations. Dymola selects the pressure in the receiver and the pressure of the ideal outlet medium in the gas cooler as iteration variables of the initialization problem after aliasing. Any other combination of a pressure at the high and a pressure at the low pressure side would be an equally good choice for the iteration variables. The system of equations cannot be solved without the

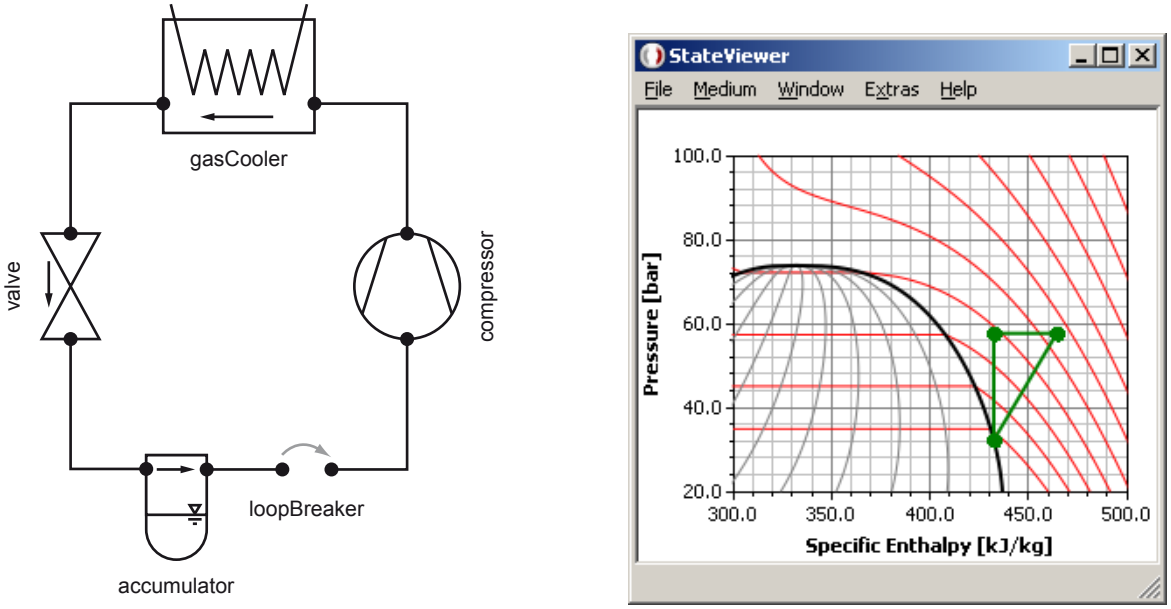


Figure 5.6: Triangular cycle modeled with component models from the `HVAC` package and representation of the numerical result in a pressure-enthalpy diagram.

specification of start values for the two pressure levels. Table 5.1 shows the three different combinations of pressures that were used as start values and motivates the selection of the specific combination.

Start Values		**Motivation**
Low Pressure	**High Pressure**	
1 bar	10 bar	Both selected pressures are below the steady-state solution yielding a solution process that starts at a too low pressure level.
32 bar	57 bar	The selected pressures are already close to the steady-state solution and yield a very fast initialization process.
100 bar	200 bar	Both selected pressures are higher than the steady-state solution yielding a solution process that starts at a too high pressure level.
1 bar	200 bar	This selection of pressures does not yield a solution.

Table 5.1: Sets of start values for the triangular cycle shown in figure 5.6.

The TILFluidsLib uses the C++ interface of TISC to send all important properties of a medium object directly after its *setState_XX()* function has been called. The StateViewer uses a specialized TISC interface to receive the sent values from the TISC simulation server and visualizes them in thermodynamic phase diagrams and in time plots where the iteration number is used as time information. The graphical visualization of the iteration process allows the user to find out which variables are used in the process and in which direction the iteration process develops. Both information can be used to specify appropriate start

values or to fine-tune existing start values to allow for a successful iteration or to speed-up the process.

Figure 5.7 shows the iteration process for the high pressure depending on the specified start values listed in table 5.1. The number of iterations is the total number of calls of *setState_XX()* functions of the TILFluidsLib during initialization. Figure 5.8 presents the same diagram for the iteration of the low pressure. As expected, the iteration process with start values of 32 bar and 57 bar for the low and the high pressure respectively requires the smallest number of calls of the TILFluidsLib. The other two sets of start values require more function calls to obtain the steady-state solution.

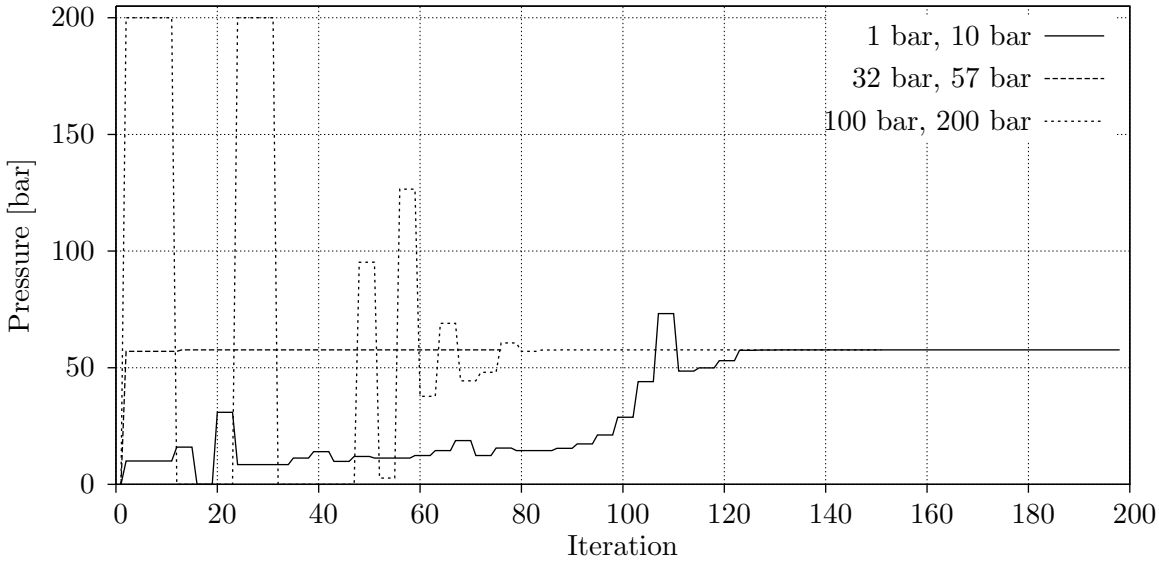


Figure 5.7: Iteration of high pressure for different start values.

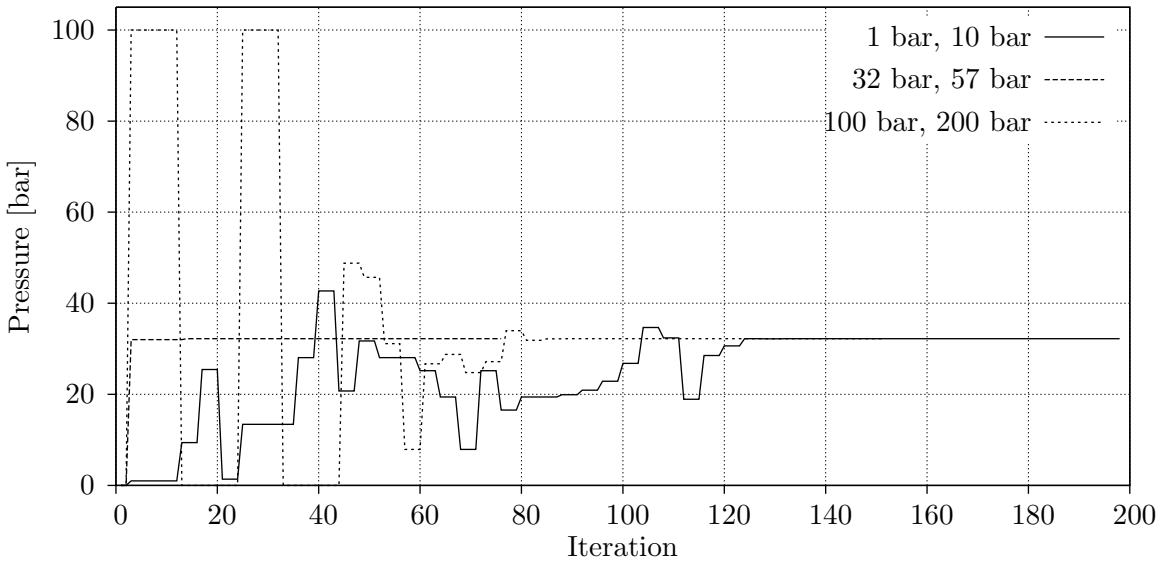


Figure 5.8: Iteration of low pressure for different start values.

Chapter 6

Thermoelectric Applications

This chapter presents a model for a Peltier water-water heat exchanger that was developed by combining existing models from the new component model library with a new model for the Peltier element and with electric component models from the Modelica Standard Library. The new model demonstrates the extendibility of the presented component model library to cover multidisciplinary thermodynamic problems. The numerical results obtained with the new heat exchanger model are compared to measurements from a prototype Peltier heat exchanger.

6.1 Introduction

Thermoelectric technology allows for the direct conversion of a temperature difference into an electric potential and vice versa. The French physicist Jean Peltier discovered in 1834 that an electric current sent through a circuit made of dissimilar conducting materials yields heat absorption at one junction and heat rejection at the other. Modern thermoelectric modules utilize doped bismuth telluride as semi-conductor to achieve optimum performance. They can act as coolers, heaters, or power generators and applications of small capacity thermoelectric modules are widespread. However, applications of large capacity thermoelectric devices have been limited in the past by the low efficiency of thermoelectric modules. Recent scientific advances regarding new materials and assembly methods for thermoelectric modules as well as the increasing concerns about fuel economy, harmful emissions of particulate matter, and chemical refrigerants revived the interest in thermoelectric technology. The inherent advantages of thermoelectric systems such as the absence of moving parts, quiet operation, and environmental friendliness of the module itself have further increased the interest. Several investigations for applications of large capacity thermoelectric modules in the fields of refrigeration and air-conditioning (Winkler et al., 2006), waste heat recovering (Zorbas et al., 2007), or superconduction (Bos et al., 1998) have been carried out with promising results.

The IfT is working on Peltier heat exchangers for new applications. The main focus of this research is an optimization of the thermoelectric efficiency of Peltier heat exchanger by developing new concepts and by improving the heat transfer as well as the proposal of control

strategies. The new component model library presented in chapter 4 was used to develop a model for a prototype Peltier water-water heat exchanger. The new model demonstrates the application and the extendibility of the new component model library as well as its multidisciplinarity. It combines existing models, a new model for the Peltier element, and electric component models from the Modelica Standard Library. The dynamic behavior of the heat exchanger model is compared to measurements from a prototype Peltier heat exchanger subjected to a sudden reversion of the applied voltage.

Section 6.2 gives a short introduction to thermoelectric refrigeration and derives the equations used to model the Peltier element. The prototype Peltier water-water heat exchanger is explained in section 6.3. Section 6.4 gives a detailed description of the new model for Peltier heat exchangers that was developed by combining existing models with a new model for a Peltier element.

6.2 Thermoelectric Refrigeration

Thermoelectric refrigeration is achieved when a direct current I is passed through one or more pairs of n-type and p-type semiconductors connected with a metal with high conductivity such as copper as sketched in figure 6.1.

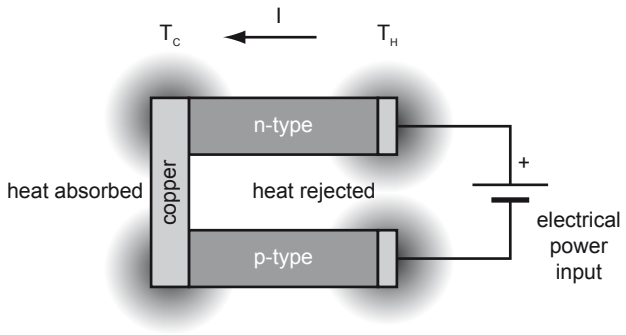


Figure 6.1: The Peltier effect (thermoelectric cooling) from Rowe (2006).

When the electric current passes from the n-type to the p-type semiconductor, electrons pass from a low energy level in the p-type material through the interconnecting conductor to a higher energy level in the n-type material. Thus the temperature T_C of the interconnecting conductor decreases and heat is absorbed from the environment. The absorbed heat is transferred by electron transport through the semiconductors to the other end of the device. It is liberated as the electrons return to a lower energy level in the p-type material yielding an increased temperature T_H.

This phenomenon is known as the Peltier effect and is described by the Peltier coefficient π defined as the product of the Seebeck coefficient α of the semiconductor material and the absolute temperature. The Peltier coefficient relates to a cooling effect when the electric current passes from the n-type to the p-type semiconductor and a heating effect when the polarity of the power supply is changed. Reversing the direction of the electric current also reverses the temperatures of the hot and cold ends.

The amount of heat absorbed at the cold end does not only depend on the product of the Peltier coefficient and the electric current flowing through the thermoelectric module but also on two other effects: Due to the temperature difference between the cold and the hot ends of the semiconductors, heat is conducted through the semiconducting material from the hot to the cold end. The amount of conducted heat depends on the thermal conductance κ of the material as well as on the temperature difference. The second effect occurs when the electric current is passing through the semiconductors. The electrical resistance R causes the generation of the so-called Joule heat in equal shares at the cold and the hot side of the thermoelectric device. The Joule heat is dependent on the electrical resistance and proportional to the square of the electric current and therefore eventually becomes the dominant factor.

The heat absorption rate at the cold side of the thermoelectric module can be described taking into account the three different effects mentioned above by

$$\dot{Q} = \alpha \, T_C \, I - \frac{1}{2} I^2 R - \kappa (T_H - T_C) \tag{6.1}$$

where α is the differential Seebeck coefficient sometimes referred to as α_{ab}, R the electrical resistance of the thermoelements in series, and κ the thermal conductance of the thermoelements in parallel. The energy efficiency of the thermoelectric device is described by its coefficient of performance (COP) defined as the net heat absorbed at the cold junction divided by the electric power input

$$COP = \frac{\dot{Q}}{P_{el}} = \frac{\alpha \, T_C \, I - \frac{1}{2} I^2 R - \kappa (T_H - T_C)}{\alpha \, \Delta T \, I + I^2 R} \tag{6.2}$$

The refrigeration capability of a semiconductor material depends on a combined effect of the Seebeck coefficient α, the electrical resistivity ρ, and the thermal conductivity κ of the material over the operational temperature range between the cold and the hot junctions. The electrical resistivity is defined as

$$\rho = R \, \frac{A}{l} \tag{6.3}$$

where A is the cross-sectional area of resistive material and l its length. The three material properties are combined in the thermoelectric figure of merit Z defined as

$$Z = \frac{\alpha^2}{\kappa \rho} \tag{6.4}$$

The figure of merit is often used by material scientists to describe semiconductor materials.

6.3 Prototype Peltier Heat Exchanger

The Peltier effect can be used for heating and cooling in practical applications by combining thermoelectric modules with conventional heat exchangers. The fluid flowing through the heat exchanger acts as a heat sink at the hot side of the thermoelectric module and as a heat source at the cold side. Figure 6.2 shows the assembly of the prototype Peltier heat exchanger used for all measurements.

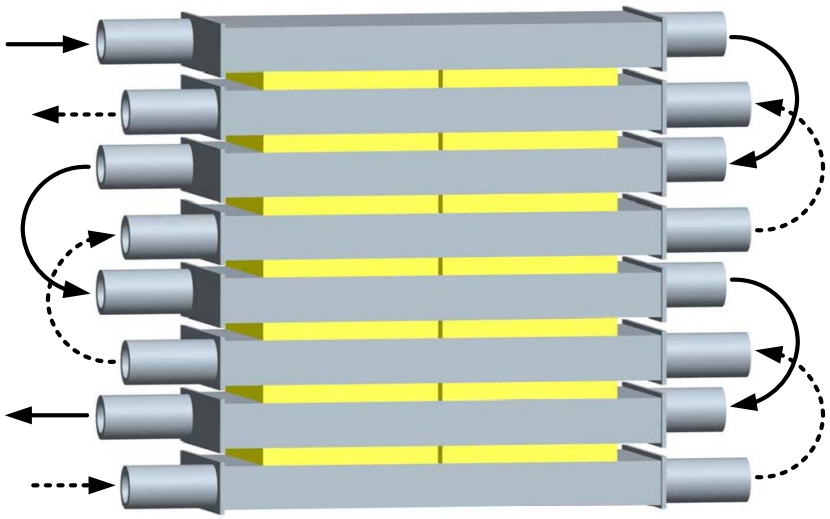


Figure 6.2: CAD drawing of the prototype Peltier water-water heat exchanger. The Peltier elements are the flat cuboids between two aluminium channels. The orientation of the Peltier elements successively changes between the rows of channels.

Because of the consolidated design and small size of the prototype heat exchanger, water was chosen as refrigerant at both sides. The heat exchanger consists of rectangular aluminium channels whose endings are covered by plates. Aluminium cores act as connecting tubes.

The prototype heat exchanger is assembled so that both sides of the thermoelectric module are in contact with a channel. The arrangement of the thermoelectric modules has to be taken into account for an efficient utilization of the Peltier effect. It is necessary to either heat or cool the channels. A combination of heating and cooling does not yield a reasonable application.

To increase the flow velocity and the heat exchange between the fluid and the wall, three barriers were installed in each channel. A Computational Fluid Dynamics (CFD) simulation was carried out to determine the flow situation in the channel. The simulation results proved that the fluid meanders through the channel and showed that fluid circulation caused by the barriers leads to a decrease in dead storage capacity and thus to an improvement in the heat exchange between fluid and wall. Figure 6.3 shows a single channel and the flow path.

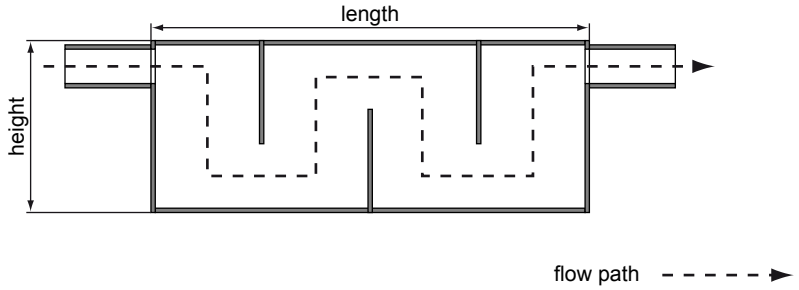


Figure 6.3: Single channel element of prototype Peltier heat exchanger.

6.4 Peltier Heat Exchanger Model

In order to model the prototype Peltier heat exchanger presented in the previous section, a model for the Peltier element had to be developed. The new component model library presented in chapter 4 does not contain any models for electrical components. Instead, the electrical component model library provided in the Modelica Standard Library as described by Clauß et al. (2002) was used as a starting point for the development of the model for the Peltier element. Figure 6.4 shows a class diagram of the developed model for a Peltier element. The material properties of the semiconductor material are stored in records extending from `BaseMaterial`. Two heat ports derived from the `HeatPort` connector defined in the new component model library and two electrical pins defined in the Modelica Standard Library are the interface of the Peltier element.

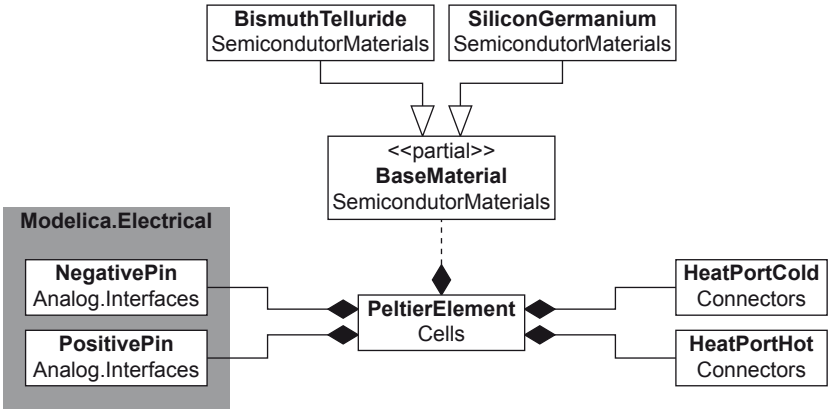


Figure 6.4: UML class diagram of `PeltierElement`.

The equations to model a Peltier element were presented in section 6.2. Based on these equations, the following set of equations is formulated in the `PeltierElement` model

$$I_{negative} + I_{positive} = 0 \tag{6.5a}$$

$$U_{positive} - U_{negative} = R I_{negative} \tag{6.5b}$$

$$P_{el} = I_{negative}(U_{positive} - U_{negative}) \tag{6.5c}$$

$$P_{el} + \dot{Q}_C + \dot{Q}_H = 0 \tag{6.5d}$$

$$COP = \frac{\alpha\, T_C\, I - \frac{1}{2} I^2 R - \kappa(T_H - T_C)}{\alpha\, \Delta T\, I + I^2 R} \tag{6.5e}$$

$$\dot{Q}_C = -COP \cdot P_{el} \tag{6.5f}$$

$$\dot{Q}_H = (1 + COP) \cdot P_{el} \tag{6.5g}$$

The `PeltierElement` model is instantiated in the `PeltierCell` model along with two models for electrical insulators as shown in figure 6.5.

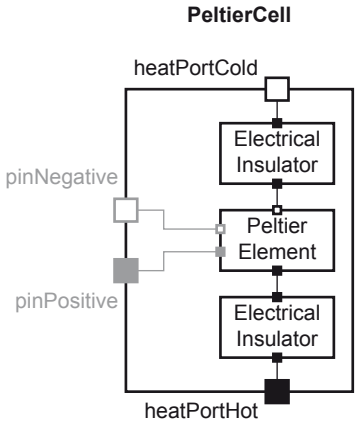


Figure 6.5: `PeltierCell` defined in `TIL_AddOn_ThermoElectrics`.

The electrical insulators prevent a short circuit between the Peltier elements and the aluminium channels. The `ElectricalInsulator` model allows for the specification of a thermal resistance R and uses the following equation to model the heat flow

$$\dot{Q}_{in} = \frac{T_{in} - T_{out}}{R} \tag{6.6}$$

where $\dot{Q}_{in}$ is the entering heat flow rate, T_{in} the temperature at the inlet port, and T_{out} the temperature at the outlet port. Note that the naming of the heat ports in figure 6.5 is chosen for the default case that is a positive electric current in the conventional current notation. The hot side eventually becomes the cool side and vice versa if the direction of the current is reversed. The swapping of the corresponding temperatures T_C and T_H is implemented using a smooth transition function as explained in section 4.6 with a very short transition period.

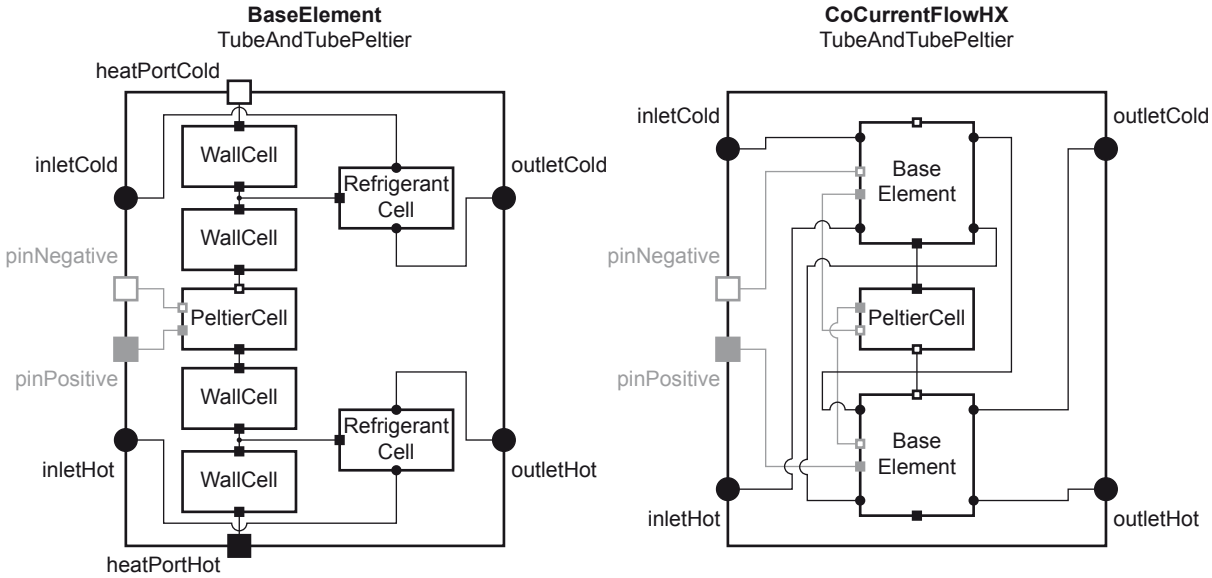


Figure 6.6: `BaseElement` and its usage in a Peltier water-water heat exchanger model from `TIL_AddOn_ThermoElectrics`. The `PeltierCell` is shown in figure 6.5.

In order to model the prototype Peltier heat exchanger shown in figure 6.2 in a flexible way, an additional model called `BaseElement` is introduced that models a single layer of the prototype Peltier heat exchanger. A layer consists of two aluminium channels as sketched in figure 6.3 and the Peltier element in between these two channels. The model is illustrated in the left picture in figure 6.6. A refrigerant cell and two wall cells from the new component model library (see section 4.7 for more information) are combined to model a single channel. The reason for using a `RefrigerantCell` instead of a `LiquidCell` is that the new model was developed to cover cases of evaporating and condensing fluids in both fluid paths. The two channels are connected using a `PeltierCell` described above. Note that the `BaseElement` model in figure 6.6 can be directly used as a single pass heat exchanger.

The model for the prototype Peltier heat exchanger assembles instances of the `BaseElement` and the `PeltierCell` model as shown in the right picture in figure 6.6. The prototype heat exchanger from figure 6.2 is for example composed of four base elements and three Peltier cells in between. Figure 6.7 shows the class diagram of the new `TubeAndTubePeltier` heat exchanger model. Note that the wall material model and all heat transfer and pressure drop models are skipped for simplicity. The complete structure would be similar to the structure of the `FinAndTube` heat exchanger model shown in figure 4.19.

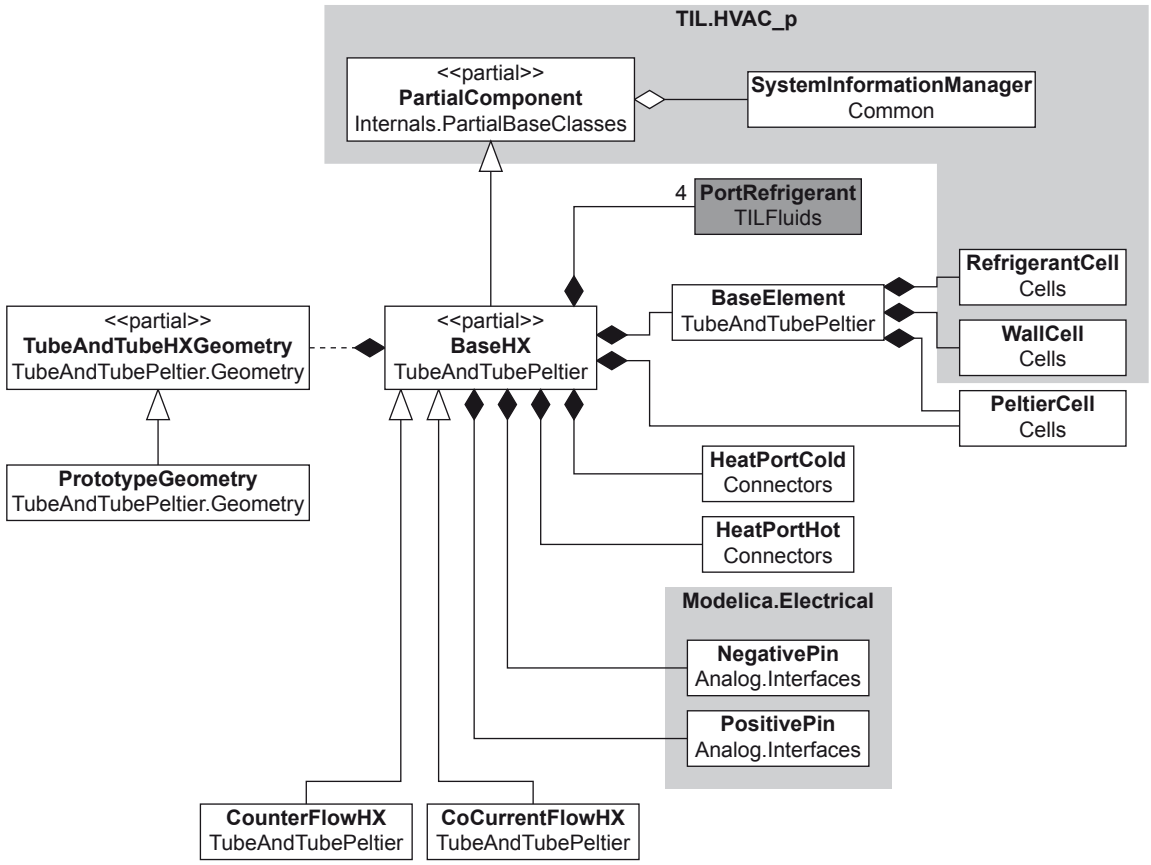


Figure 6.7: UML class diagram of `TubeAndTubePeltier` heat exchanger in `TIL_AddOn_ThermoElectrics`. The wall material model and all heat transfer and pressure drop models are skipped for simplicity. Class diagrams of the cells can be found in figures 4.13, 4.16, and 6.4.

6.5 Measurements

A series of measurements was carried out with the prototype Peltier water-water heat exchanger presented in section 6.3. Figure 6.8 shows a schematic diagram of the test stand used for all measurements.

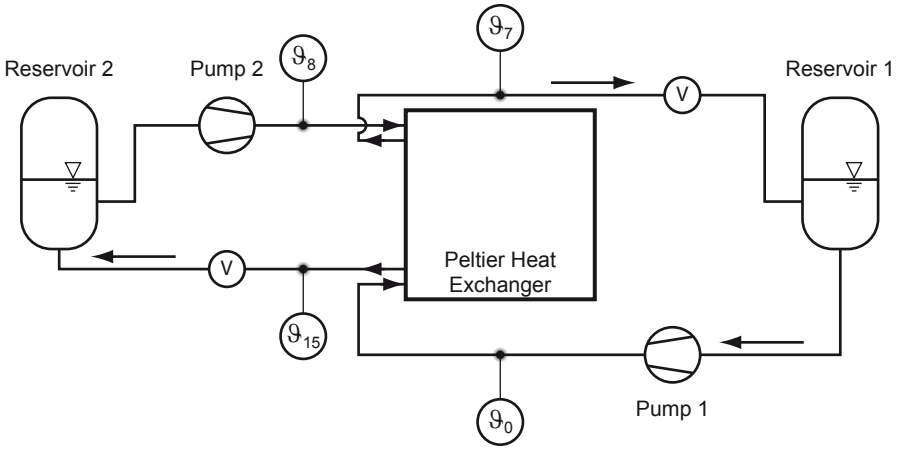


Figure 6.8: Schematic diagram of Peltier heat exchanger test stand.

To ensure a constant temperature at the water inlet of the prototype, a reservoir was used in both cycles. Water was pumped from the reservoirs into the prototype and flowed back after running through the heat exchanger. The volume flow rates were regulated with appropriate throttling devices and measured by using conventional water meters.

Besides the volume flow rates characteristic parameters such as the water temperatures at the inlet and outlet of each aluminium tube or the electric current and voltage drop over every Peltier element were taken up. The boundary conditions for the measurements were selected in consideration of showing the applicability of the simulation for different premises. Therefore a low, a medium, and a high water inlet temperature were chosen and each condition measured by using a low and a high volume flow rate respectively. Each measurement was carried out at a working-voltage of 10 V. A summary of the boundary conditions for all measurements is given in table 6.1.

	Water Stream 1		**Water Stream 2**	
#	$\dot{V}_1$ [l/min]	ϑ_0 [°C]	$\dot{V}_2$ [l/min]	ϑ_8 [°C]
1	2.05	4.00	2.00	4.00
2	0.90	4.00	0.85	4.00
3	2.20	18.00	2.10	18.00
4	0.85	18.00	0.80	18.00
5	2.35	30.00	2.40	30.00
6	1.00	30.00	1.10	30.00

Table 6.1: Measurements with prototype Peltier water-water heat exchanger.

All measurements were carried out in the same way: After reaching a stationary point for the boundary conditions listed in table 6.1, the direction of the electric current was changed from positive to negative in the conventional current notation. The resulting change in temperature was detected until the values became stationary again.

An evaluation of the quality of the measurements was carried out by comparing the sum of the input power and the gained cooling capacity to the achieved heating capacity according to

$$P_{el} + \dot{Q}_{cooling} = \dot{Q}_{heating} \tag{6.7}$$

The cooling capacity as well as the heating capacity was calculated from

$$\dot{Q} = \dot{m} c_p \Delta T \tag{6.8}$$

and the electric power from

$$P_{el} = IU \tag{6.9}$$

The deviation within the balance has to be zero for the ideal case. The deviation of the two balances for each measurement is shown in figure 6.9. It can be seen that the deviation lies between 1% and 8%, and that the average value lies around 4%. A connection between the direction of the electric current and the resulting deviation could not be identified.

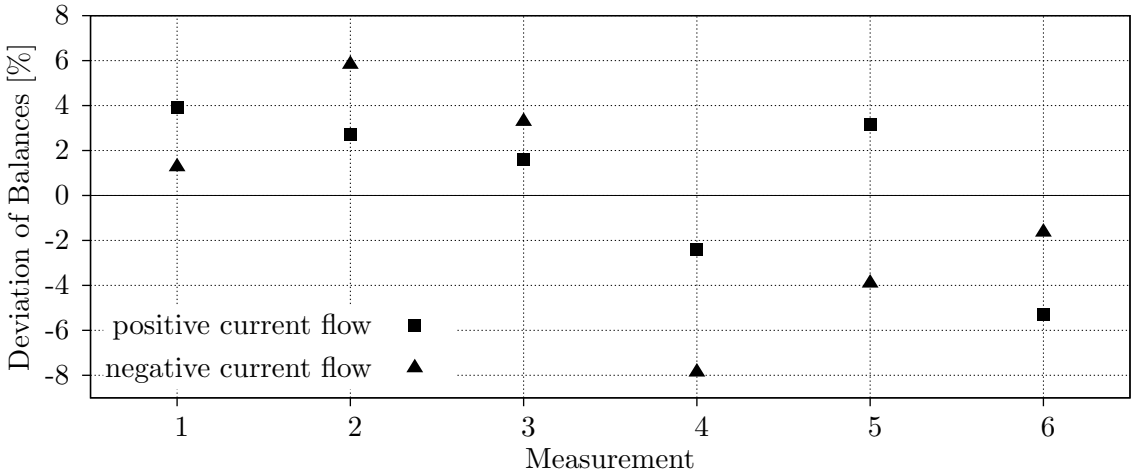


Figure 6.9: Deviation of electrical and thermal balances for all measurement points.

To exclude the existence of a statistical error and to confirm that the deviations of the balances are lying within the measuring accuracy, an error analysis was carried out. Therefore, Gauss' error propagation law was used according to

$$\Delta\overline{F} = \sqrt{\left(\frac{\partial F}{\partial x}\Delta\overline{x}\right)^2 + \left(\frac{\partial F}{\partial y}\Delta\overline{y}\right)^2 + \dots} \tag{6.10}$$

Measurement 4 from table 6.1 was selected for an exemplarily error analysis. A variation of relevant measurands was carried out to find out the impact of these measurands on the total error and to identify possible potentials for further optimization.

Figure 6.10 shows the impact of the error occurring during the measurement of the temperature difference $\Delta\Delta T$ between the inlet and outlet of the Peltier prototype heat exchanger and during the estimation of the volume flow rate $\Delta\dot{V}$ on the resultant heating or cooling capacity.

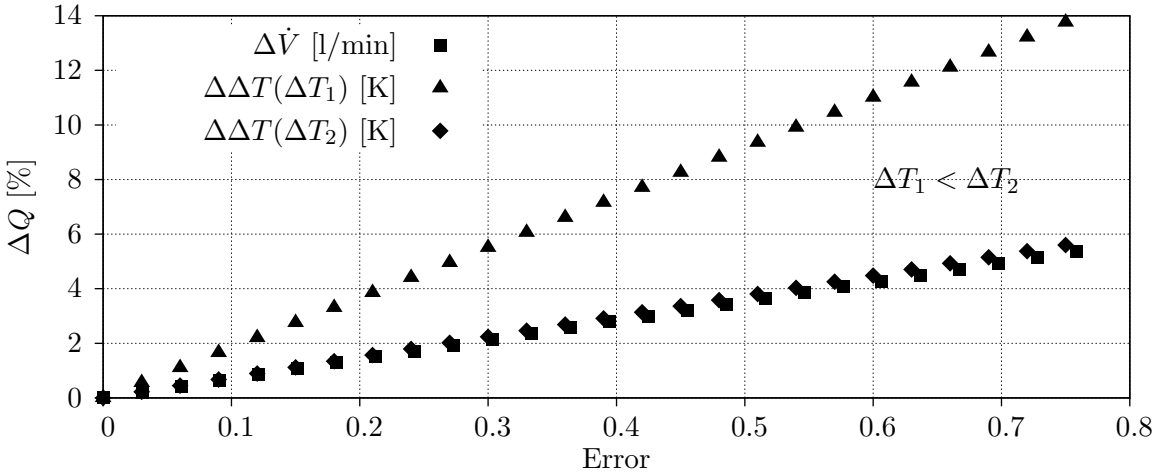


Figure 6.10: Error for measurement 4 from table 6.1. The corresponding units are given in the key.

Due to the fact that the measuring accuracy of a thermocouple lies at about 0.3 K, the maximum error for the mathematical calculation of the temperature difference can be expected to be 0.6 K when using temperatures measured with two independent thermocouples. This error can be reduced to 0.1 K if the temperature difference is measured using two thermocouples connected in series which was done for all measurements presented in table 6.1.

In consideration of the volume flow rate measurements, the deviation of the values estimated with conventional flow meters and the actual values lies between 4% and 9% which results in a maximum deviation of 0.09 l/min. The concluding summation yields - under consideration of these conditions - the result that even the measurements with a deviation of balances of 8% are lying within measuring accuracy.

6.6 Numerical Results

Simulations were carried out for all measurements listed in table 6.1. A complete system setup in Dymola is shown in figure E.2 in appendix E.2. Measurement values were used for the electric current, for two volume flow rates, and for the water temperatures ϑ_0 and ϑ_8 at the two heat exchanger inlets. The Peltier modules used in the prototype Peltier heat exchanger are standard low-cost bismuth telluride modules without any further specification from the manufacturer. Constant properties for the Seebeck coefficient α and the thermal conductance κ taken from Rowe (2006, table 9.1) were used in the Peltier element model. The electrical resistance R of the thermoelectric module was not specified by the manufacturer and had to be determined from the measurements. The reversion of the applied voltage was implemented using a smooth transition function with a period of $\Delta t = 1$ s. This section

describes the results obtained for the simulation of measurement 4 from table 6.1. A constant coefficient of heat transfer $\alpha = 4,100$ W/(m^2 K) was used. This coefficient of heat transfer was determined based on a CFD simulation of the flow through a single aluminium channel as presented in appendix E.2.

Figure 6.11 shows the temperature distribution in the prototype Peltier heat exchanger before and after reversion of the applied voltage. The numbering of the water streams and of the walls refers to the numbering of the two independent water circuits as presented in figure 6.8. The water temperatures are shown for the inlet of each channel and for the outlet of the last channel for both water streams. The wall temperatures are averages of the temperatures in the center of both wall cells connected to the same refrigerant cell as shown in figure 6.6.

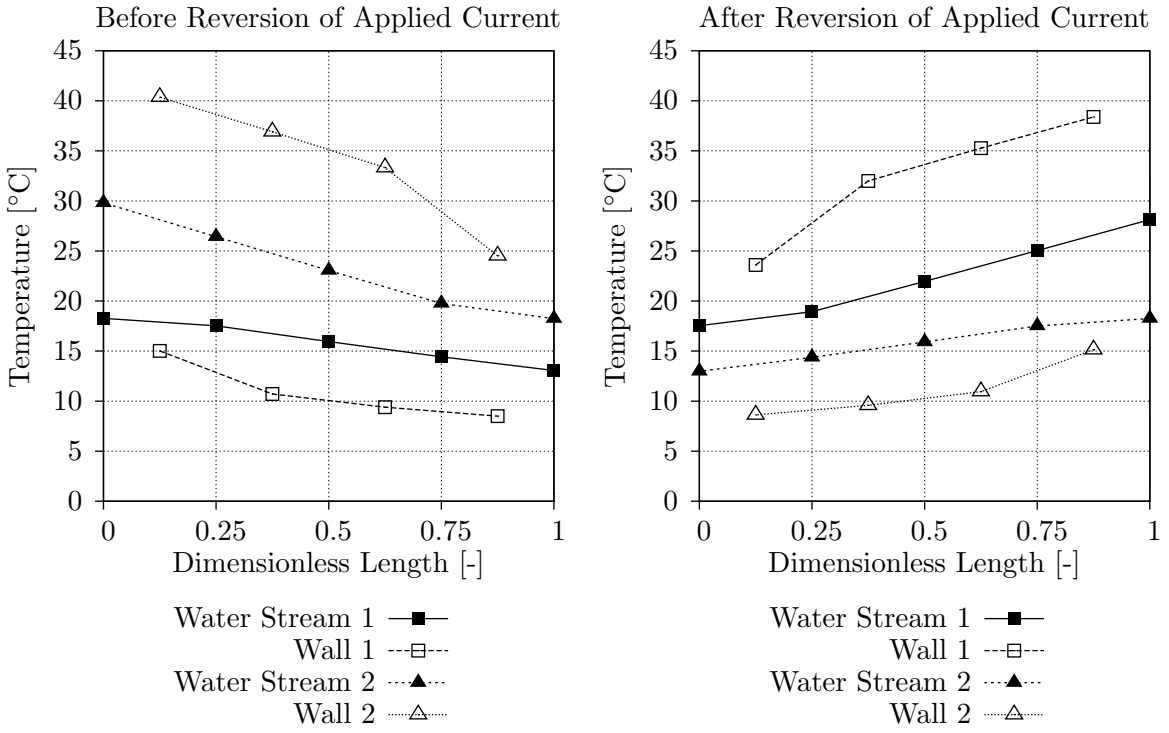


Figure 6.11: Temperature distribution in prototype Peltier heat exchanger before and after reversion of the applied voltage for measurement 4 from table 6.1.

Figure 6.11 shows that the temperature change in the entrance channel of each water stream is smaller than in all other subsequent channels. This is caused by the fact that the entrance channels are insulated at one side and connected to a Peltier element at the other side whereas all other channels are connected to a Peltier element at both sides. The two diagrams shown in figure 6.11 are mirror-symmetrical which demonstrates the reversibility of the process.

Figure 6.12 shows a comparison of the measured outlet temperature for each water stream with the values obtained from the transient simulation. The top picture shows the change in the electric current I caused by the reversion of the applied voltage.

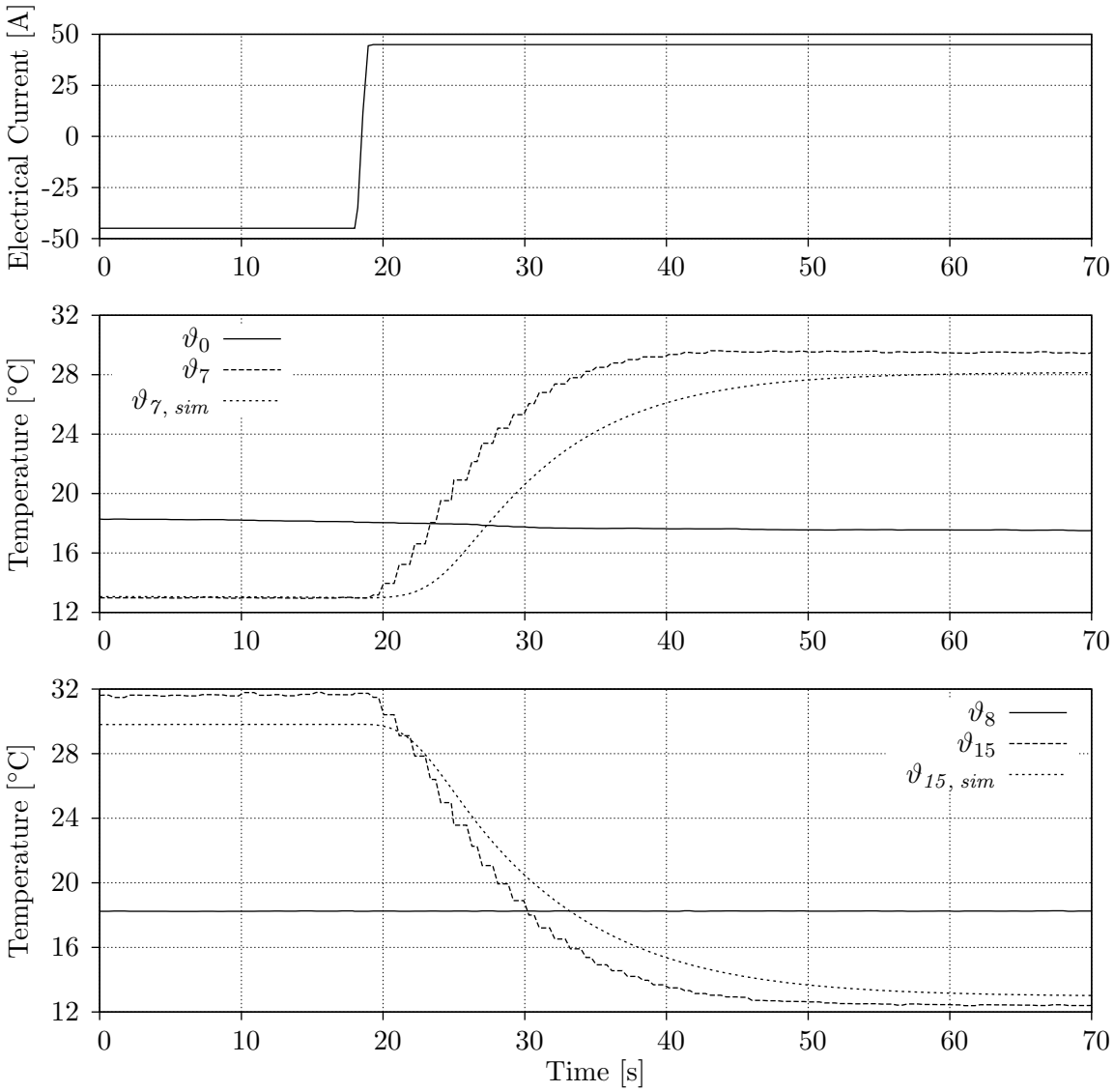


Figure 6.12: Measured and simulated water temperatures at inlets and outlets of prototype Peltier heat exchanger for measurement 4 from table 6.1.

Figure 6.12 illustrates that the simulated start and end temperatures differ from the measured temperatures. The simulated system also reacts slower to the sudden reversal of the applied voltage than the real system. Further measurements are required to improve the model of the Peltier element that is currently based on material constants taken from the literature and the measured electrical resistance as explained in the beginning of this section.

The presented model for a prototype Peltier water-water heat exchanger was used successfully in transient simulations. It was shown that the new component model library presented in chapter 4 can be extended with new models and with models form other existing component model libraries to allow for the simulation of multidisciplinary systems. The presented model can be extended to cover other Peltier heat exchangers. A very interesting alternative concept to be analyzed in the future is a refrigerant-air heat exchanger with Peltier modules in between.

Chapter 7

Ejector Refrigeration System

This chapter presents a model for an ejector that was developed to analyze the achievable coefficient of performance improvements when using an ejector as a throttling device in a CO_2 refrigeration system. The new ejector model was developed based on measurement results from a prototype ejector partly developed within the scope of this thesis. With the new ejector model, a comparison between a conventional CO_2 refrigeration system and an ejector CO_2 refrigeration system both featuring an internal heat exchanger was carried out using steady-state component models from the new component model library. It could be shown that an ejector yields an increased efficiency for those kind of systems.

7.1 Introduction

The pressure difference between the high and low pressure part of a CO_2 refrigeration system is relatively high compared to other refrigerants (e.g., R134a) due to the transcritical cycle design. Lorentzen (1983) was one of the first to realize that this high pressure difference causes significant throttling losses. Using an ejector instead of the throttling valve is one very promising way to recover parts of the otherwise lost kinetic energy. The main advantage of an ejector compared to other expansion devices is its simple design without any moving parts thus not requiring any maintenance. Many researchers worldwide, such as Li and Groll (2006), Elbel and Hrnjak (2006a), Drescher et al. (2007), and Ozaki et al. (2004), are working on ejectors for CO_2 refrigeration cycles. The pure number of projects carried out in this field already indicates two things: the high power saving potential of this new system design as well as the many problems that arise from designing a device to mix two two-phase flows at high velocities. The Japanese company Denso was the first company that introduced ejectors in commercial CO_2 systems in 2003 (DENSO Cooperation, 2003). The first application was in a heat-pump system used to reheat bath water allowing an entire family to enjoy a bath using the same water which is very common in Japan.

This chapter first gives a general introduction to ejector cycles and to the history of CO_2 as a refrigerant in sections 7.2 and 7.3 respectively. The state of the art for CO_2 ejectors

is summarized in section 7.4. The test stand and the prototype ejector designed and built at IfT partly within the scope of this thesis are described in section 7.5. The results from measurements presented in section 7.7 were used to determine the parameters required for the ejector model described in section 7.6. The model was then used to study the COP (coefficient of performance) for various boundary conditions. Some of the numerical results are presented in section 7.8.

7.2 Ejector Refrigeration Systems

An ejector is composed of the driving nozzle, the suction nozzle, the mixing section, and the diffuser as illustrated in figure 7.1. The driving flow is accelerated in the primary nozzle and enters the mixing section at a pressure lower than the pressure at the suction inlet. The pressure difference between the inlet and the outlet of the suction nozzle yields an acceleration of the suction flow. Both flows are mixed in the mixing section. The diffuser reduces the velocity of the mixed CO_2 stream resulting in a pressure increase.

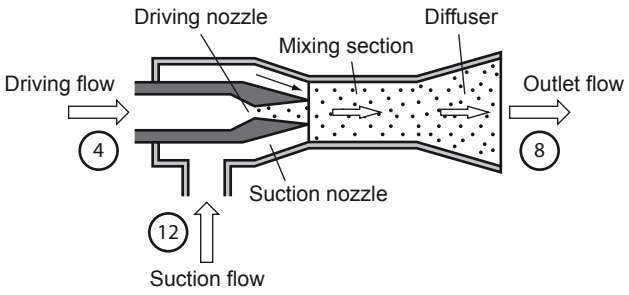


Figure 7.1: Schematic cross sectional view of a CO_2 ejector.

Figure 7.2 shows a schematic diagram of a CO_2 ejector refrigeration system. Three components are different compared to a conventional refrigeration system. The ejector replaces the throttling valve, a separator is used instead of the accumulator, and an additional control valve is introduced at the liquid outlet of the separator. The pressure-enthalpy diagram of a CO_2 ejector refrigeration system is shown in figure 7.3 using the same numbering scheme as in figure 7.2.

Three different pressure levels exist in the ejector refrigeration system: The high pressure level p_h, the intermediate pressure level p_{im}, and the evaporation pressure level p_{evap}. The separator splits the refrigerant flow into a vapor stream (1) and a liquid stream (10). The vapor stream is compressed from the intermediate to the high pressure level in the compressor (2). The high pressure gas stream then flows through the gas cooler where it is cooled to a lower temperature (4) before it enters the ejector through the driving inlet. The liquid stream leaving the separator (10) is flowing through a control valve where it is expanded from the intermediate to the evaporation pressure level (11). It is evaporated in the evaporator and enters the ejector through the suction inlet (12). The point (12) can be as shown in the two-phase region but it might also be superheated. The driving flow is leaving the driving nozzle at (5) and is mixed with the suction flow from the suction nozzle (6) as shown in

figure 7.1. The mixed stream then enters the diffuser in (7) and is decelerated yielding an increase in pressure. It leaves the ejector at (8) and is again split into a vapor and a liquid stream in the separator. Note that the points (3) and (9) are skipped on purpose for the case that an internal heat exchanger is added to the system.

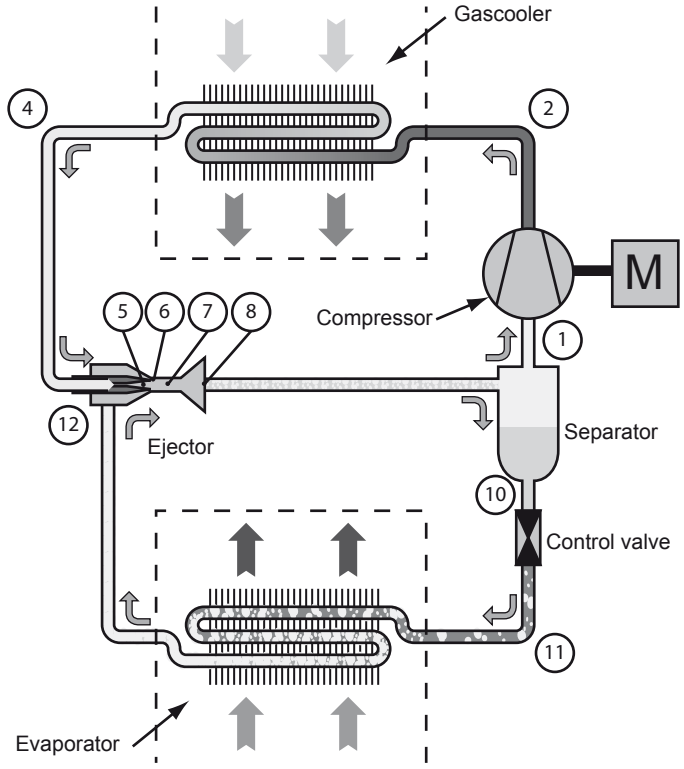


Figure 7.2: Schematic diagram of a CO_2 ejector refrigeration system.

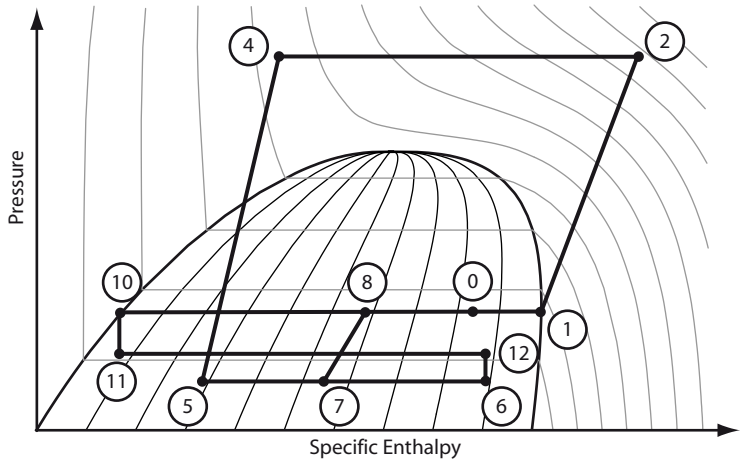


Figure 7.3: Pressure-enthalpy diagram of a CO_2 ejector refrigeration system.

7.3 A Brief History of CO_2 as Refrigerant

Carbon dioxide is a natural refrigerant that was first used in the 19th century and the beginning of the 20th century in various air-conditioning and refrigeration applications. In Europe, CO_2 machines were often the only choice due to the legal restrictions on the use of toxic or inflammable refrigerants like NH_3, SO_2, and hydrocarbons. A good description of the rise and fall of CO_2 systems can be found in an article by Bodinus (1999). However, CO_2 was replaced in the 1930s by CFCs (chlorofluorocarbons) that were discovered as refrigerants by Thomas Midgley who had developed the tetra-ethyl lead additive to gasoline before and who "had more impact on the atmosphere than any other single organism in Earth history" (McNeill and Kennedy, 2001).

In the early 1980s, the decrease in the ozone layer was predicted to be roughly 7% over a sixty-year period. It was not before the famous article by Farman et al. (1985) that scientists as well as the public realized that there was a substantial hole in ozone layer. The especially rapid ozone depletion in Antarctica had previously been dismissed as measurement error. The stratospheric ozone depletion started to worry many and led to the Montreal Protocol, an international agreement that defines a global schedule for the phase-out of CFCs.

The CFCs in many applications were replaced by HCFCs (hydrochlorofluorocarbons) and HFCs (hydrofluorocarbons). These substances have no ozone depletion level but are thought to contribute to anthropogenic global warming due to their relatively high GWP (global warming potential) values. The regulation (EC) No 842/2006 of the European Parliament and the Council that implements the Kyoto Protocol defines a phase-out schedule of HFCs with a GWP higher than 150, especially of the widely used R134a with a GWP of about 1,300, for all mobile air-conditioning systems. The natural coolant CO_2 with a GWP of one by definition is a very promising replacement for R134a in these systems and was recently selected by the German automotive manufacturers as future refrigerant.

Lorentzen (1993) was the first to rediscover CO_2 for air-conditioning applications resulting in a 1989 international patent application in which he devised a transcritical CO_2 refrigeration system with a throttling valve that controls the high-side pressure. Over the last two decades, many researchers worked on CO_2 refrigeration and air-conditioning systems.

7.4 State of the Art

Ejectors are used to create a vacuum pressure in steam turbine exhaust condensers in thermal power stations, for the bulk handling of grains, to pump turbid water, and in many other applications. Gay (1931) was the first one to place an ejector between the evaporator and the separator in a refrigeration system. Many researchers worked on ejectors for refrigeration systems. Kornhauser (1990) and Domanski (1990) showed that cycle efficiency improvements of about 20% are theoretically possible for most HCFCs and PFCs (perfluorocarbons). Menegay (1997) measured an increase in cycle efficiency of 4% for R12. Junior (2006) compares different concepts to improve the efficiency of CO_2 refrigeration systems based on idealized system simulations in EES and reports COP improvements of up to 20% for ejector systems.

Several research groups worldwide are working on ejectors for CO_2 refrigeration and air-conditioning systems. Elbel and Hrnjak (2006b) designed and built an ejector and reported maximal COP improvements of 44%. Li and Groll (2005) also present an ejector prototype and reports COP improvements of up to 16%. Bou Lawz Ksayer (2007) presents an ejector for CO_2 systems and report COP improvements of 8 to 14%. The ejector prototype designed and built at the IfT is based on the prototype design of Elbel and Hrnjak with some modifications. Figure 7.7 shows a picture and an exploded view of the prototype ejector.

7.5 Layout of the Test Stand

An ejector test stand and a prototype ejector were designed and built at the IfT. The test stand was designed based on system simulations carried out with steady-state models from the `HVAC` package of the new component model library as presented by Richter et al. (2006). The ejector was described by using the efficiency-based ejector model presented by Kornhauser (1990). Figure 7.4 shows a schematic diagram of the ejector test stand that allows to test ejectors and ejector cycles with boundary conditions and cooling capacities common for car air-conditioning systems or small heat-pumps for residential buildings. The gas cooler is using water as secondary fluid whereas the evaporator uses a mixture of water and glycol to prevent freezing. The cooling capacity of the gas cooler is 8kW.

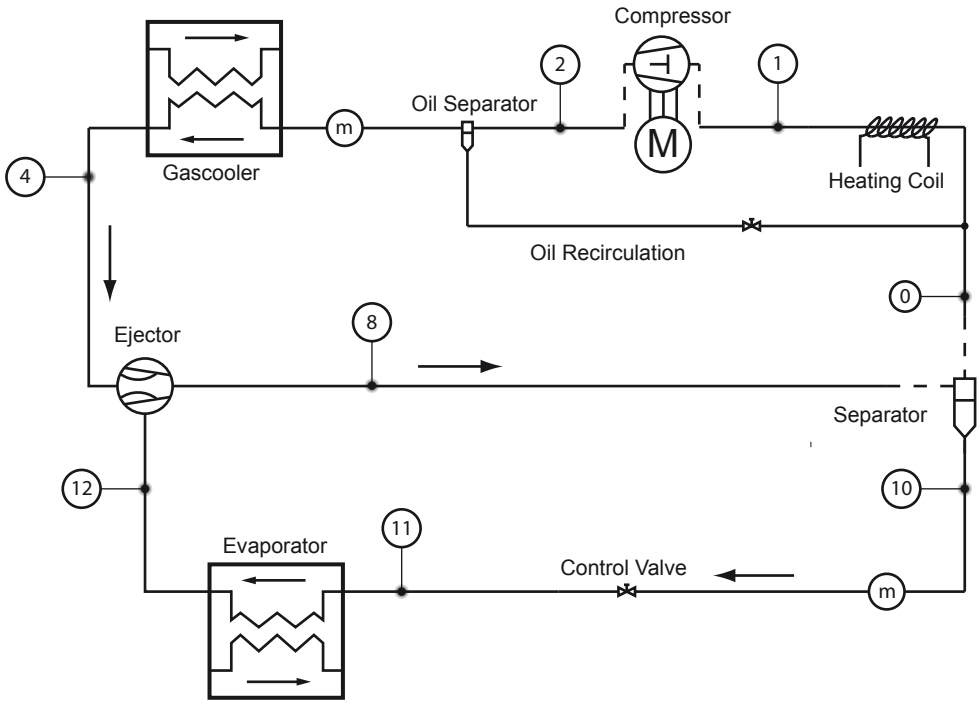


Figure 7.4: Schematic diagram of ejector test stand.

The black dots in figure 7.4 mark the points where pressure and temperature are measured. The mass flow rates in the driving and in the suction loop of the ejector cycle are measured using Coriolis mass flow meters. The liquid mass flow leaving the separator is subcooled to

avoid bubbling which would affect the accuracy of the mass flow meter. Note that this is not shown in the pressure-enthalpy diagram in figure 7.3. An electric heating at the vapor outlet of the separator protects the compressor from sucking in two-phase CO_2. The accumulated mass in the separator is determined using a scale. The test stand was designed to allow for operating points with overfilled separator. A heating coil is used in those operating points and evaporates the remaining liquid fraction of the driving mass flow from (0) to (1) which is also shown in the pressure-enthalpy diagram in figure 7.3. An oil recirculation is used to ensure that the oil is returned to the compressor.

7.6 Ejector Model

The ejector model is based on the definition of the ejector efficiency η_e which is defined independently from other components and that can be computed from simple measurements and was presented at the VDA Alternative Refrigerant Winter Meeting 2007 (Köhler et al., 2007). Similar efficiencies are defined by Kornhauser (1990), Matsuo et al. (1982), and Elbel and Hrnjak (2006b). The ejector model is based on a black box model as shown in figure 7.5.

The black box contains a compressor and a turbine that are mounted on a common shaft. In the black box ejector model, the driving mass flow $\dot{m}_D$ is expanded from the high pressure level p_h to the intermediate pressure level p_{im}. The expansion energy is used to compress the suction mass flow $\dot{m}_S$ from the evaporation pressure level p_{evap} to the intermediate pressure level p_{im}. Both processes are shown in the pressure-enthalpy diagram in figure 7.5 where the drawn through lines correspond to the ideal processes and the dashed lines to the real processes.

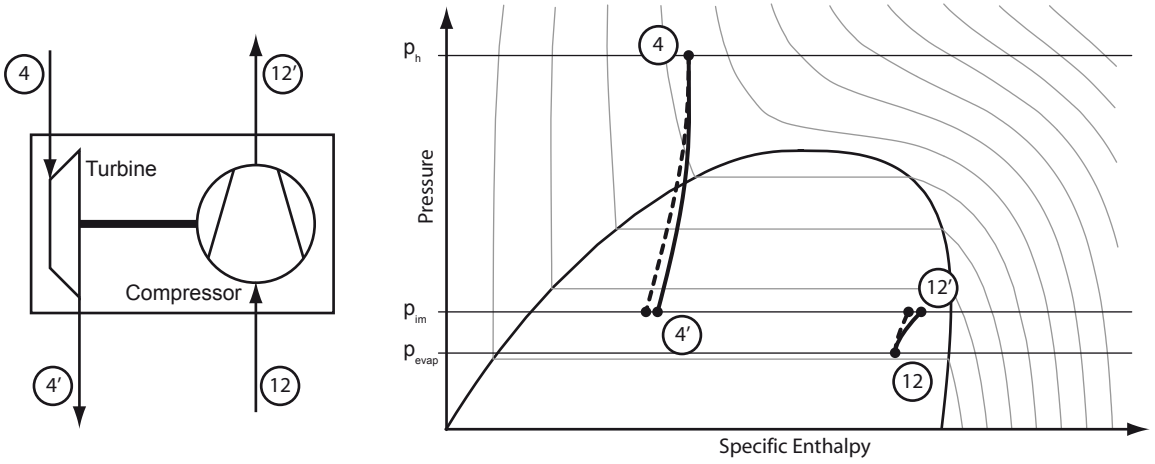


Figure 7.5: Schematic sketch of black box ejector model and corresponding pressure-enthalpy diagram.

The isentropic efficiencies of the compressor η_c and the turbine η_t can be computed as

$$\eta_c = \frac{h'_{12\,isen} - h_{12}}{h'_{12} - h_{12}} \tag{7.1}$$

$$\eta_t = \frac{h_4 - h'_4}{h_4 - h'_{4\,isen}} \tag{7.2}$$

respectively and depend on the specific enthalpies at the inlet and outlet of the compressor and the turbine. $h'_{12\,isen}$ and $h'_{4\,isen}$ are the specific enthalpies at the outlet of the compressor and turbine if those components work isentropic.

The ejector efficiency η_e is defined as the product of the two single component efficiencies and can be computed from

$$\eta_e = \eta_c \cdot \eta_t = \frac{h'_{12\,isen} - h_{12}}{h'_{12} - h_{12}} \cdot \frac{h_4 - h'_4}{h_4 - h'_{4\,isen}} \tag{7.3}$$

From an energy balance of the black box ejector model follows that

$$\frac{\dot{m}_S}{\dot{m}_D} = \frac{h_4 - h'_4}{h'_{12} - h_{12}} = \psi \tag{7.4}$$

where ψ is the ratio of suction to driving mass flow rate. Inserting this equation in equation (7.3) yields

$$\eta_e = \frac{\dot{m}_S}{\dot{m}_D} \cdot \frac{h'_{12\,isen} - h_{12}}{h_4 - h'_{4\,isen}} \tag{7.5}$$

Figure 7.5 illustrates that the enthalpy differences in the definition of the ejector efficiency are strongly influenced by the pressure differences between high and intermediate pressure and between evaporation and intermediate pressure. The higher the pressure raise within the ejector and the higher is the suction mass flow rate, the higher the ejector efficiency.

The ejector efficiency is an important characteristic of an ejector but is not sufficient to describe it entirely. The second important characteristic is the shape of the primary nozzle determining the driving mass flow rate. The converging shape of the primary nozzle of the prototype ejector suggests that a simple flow correlation can be used for its description. The most important geometrical parameter affecting the mass flow rate through a nozzle is its smallest flow area A_0. Figure 7.6 shows the flow through a nozzle. The entering refrigerant stream is accelerated in the converging part of the nozzle. The effective flow area A_{eff} is smaller than the geometrically smallest flow area A_0.

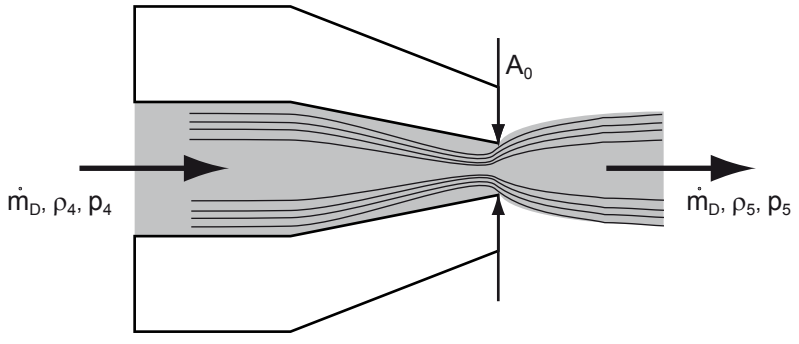


Figure 7.6: Schematic sketch of flow through a nozzle.

Wein (2002) showed that the throttling of subcooled fluids in short pipes into the two-phase region does not yield a phase change due to the short time span of the process. This implies that Bernoulli's equation for single-phase flow can be used to compute the mass flow rate through the nozzle

$$\dot{m}_D = A_{\mathit{eff}} \sqrt{2\varrho_4(p_h - p_5)} \tag{7.6}$$

where p_5 is the mixing pressure (see figure 7.3). The effective flow area A_{eff} can be computed from the geometric flow area A_0 using the following relation

$$A_{\mathit{eff}} = \alpha\, \varepsilon\, A_0 \tag{7.7}$$

where the flow coefficient α accounts for the effects of flow contraction whereas the expansion coefficient ε takes compressibility effects into account.

The ejector efficiency η_e and the flow coefficient α of the prototype ejector were determined in two series of measurements presented in section 7.7. The simple ejector model, implemented in an add-on library to the new component model library presented in chapter 4, uses equations (7.5) and (7.6) to describe the ejector. The separator model is based on the accumulator model presented in section 4.4 with an additional liquid outlet. Section 7.8 presents some numerical results obtained from the ejector model.

7.7 Measurements

A series of measurements at different operating points was carried out to determine the ejector efficiency for the ejector prototype shown in figure 7.7.

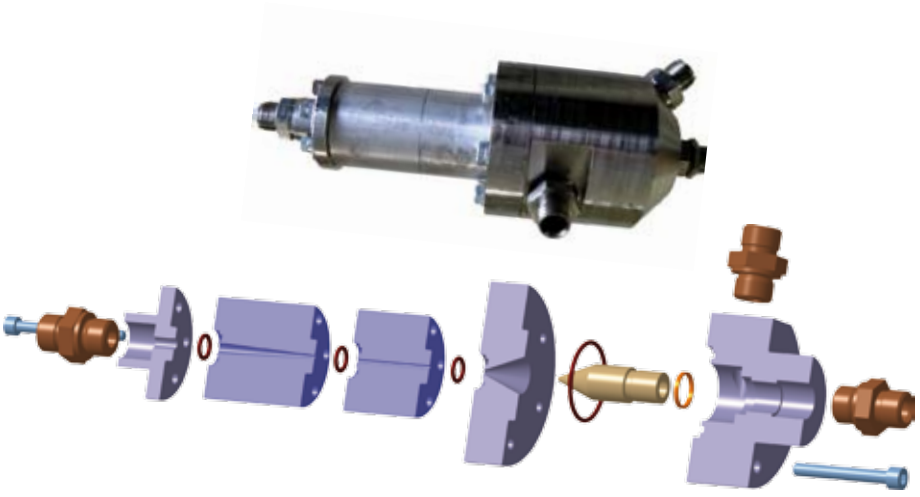


Figure 7.7: Prototype ejector designed and built at IfT.

The measurements covered different combinations of high pressures and evaporation pressures as listed in table 7.1. The ejector efficiency η_e was determined for each measurement according to equation (7.3).

	Point 4		Point 12	Derived Quantities	
#	p [bar]	T [°C]	p [bar]	ψ [-]	η_e [%]
31	80.4	35.2	32.8	0.317	10.97
32	85.4	37.2	32.4	0.321	12.67
33	90.0	42.5	30.8	0.375	12.63
34	95.1	45.4	30.1	0.363	13.44
35	100.6	48.2	29.4	0.349	13.59
36	79.6	36.8	31.0	0.240	8.69
37	85.0	39.2	30.8	0.263	10.48
38	79.8	38.5	32.9	0.398	15.70
39	84.9	40.4	32.6	0.406	16.55
40	90.0	39.5	34.2	0.404	17.46
41	94.9	40.9	33.9	0.388	17.32
42	80.0	36.8	35.4	0.502	15.66
43	84.8	38.9	34.7	0.488	17.23
44	89.4	41.3	34.0	0.479	18.88

Table 7.1: Measurements with prototype ejector to determine its efficiency η_e.

Figure 7.8 shows the computed ejector efficiencies against the mass flow ratio ψ. The fitted curve represents the relation that was implemented in the new ejector model to describe the correlation between the mass flow ratio ψ and the ejector efficiency η_e for the prototype ejector. Note that a constant ejector efficiency of $\eta_e = 20\%$ was used in the model for mass flow ratios $\psi \geq 0.54$.

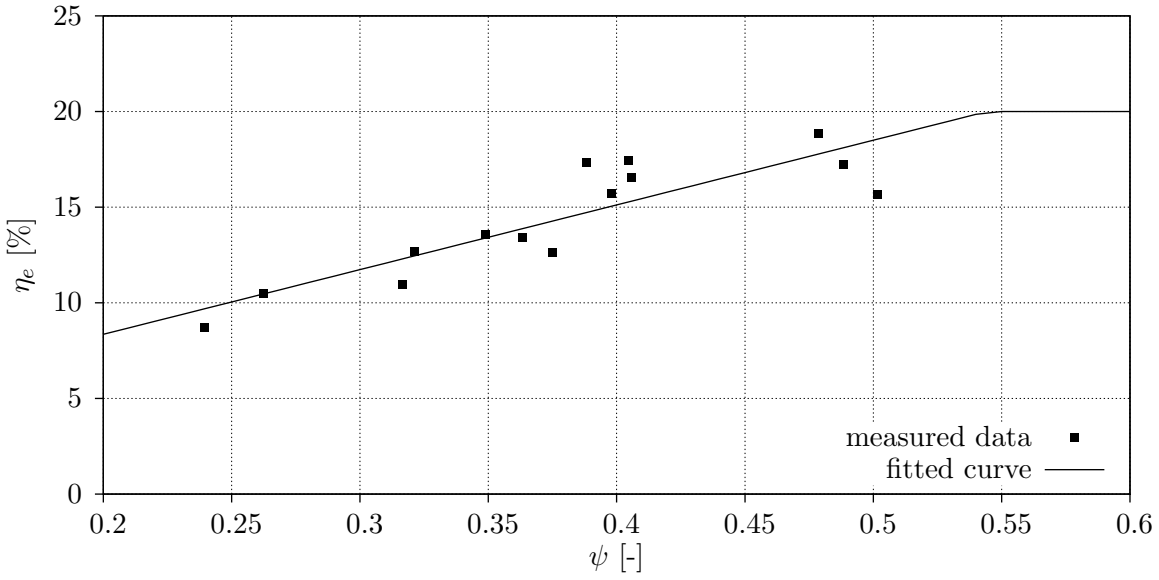


Figure 7.8: Ejector efficiency η_e from measurements listed in table 7.1.

The ejector efficiency η_e describes the amount of kinetic energy that the ejector recovers. The relatively low values of this efficiency that were achieved with the prototype ejector

indicate a large potential for further optimizations of the ejector design. This is one of the major objectives of the current research at IfT. System simulations can support this research by allowing for an evaluation of new system designs with respect to the theoretically attainable efficiency improvements of the complete cycle.

Another series of measurements with two different gas cooler outlet temperatures and varying high pressures was carried out to determine whether Bernoulli's equation can be used to describe the flow through the primary nozzle. Table 7.2 lists these measurements.

Gas Cooler Outlet Temperature $\vartheta_4 = 20$ °C

#	p_4 [bar]	p_{12} [bar]	α [-]
45	89.9	39.7	0.94
46	89.8	40.0	0.93
47	99.9	40.3	0.93
48	99.8	40.3	0.93
49	79.5	39.6	0.91
50	80.2	40.1	0.94
59	89.7	40.1	0.95
60	79.6	38.9	0.97
61	79.7	39.8	0.95

Gas Cooler Outlet Temperature $\vartheta_4 = 30$ °C

#	p_4 [bar]	p_{12} [bar]	α [-]
51	89.6	39.9	0.78
52	79.9	40.1	0.69
53	80.4	40.3	0.70
54	90.0	40.2	0.78
55	110.3	40.3	0.87
56	109.7	40.1	0.87
57	100.1	39.9	0.85
58	100.2	40.0	0.84

Table 7.2: Measurements with prototype ejector to determine the flow coefficient α.

Figure 7.9 shows the computed flow coefficients α over the high pressure p_4. A constant pressure difference of $\Delta p = 2$ bar was assumed between the evaporation pressure p_{12} and the mixing pressure p_5 for the computation of the flow coefficient α. The prototype ejector will be modified to allow for a direct measuring of the mixing pressure to test this assumption and to allow for a more precise computation of the flow coefficient α. Furthermore a constant expansion coefficient $\varepsilon = 1$ was assumed for the computation of the flow coefficient α.

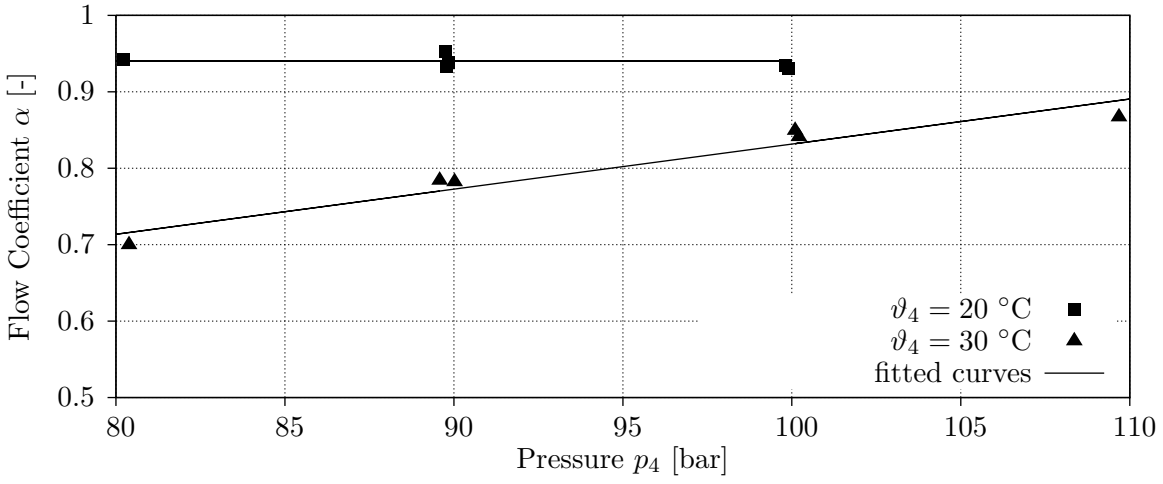


Figure 7.9: Flow coefficient α from measurements listed in table 7.1.

The results in figure 7.9 show that the flow coefficient α can be assumed to be constant for a gas cooler outlet temperature $\vartheta_4 = 20$ °C but not for $\vartheta_4 = 30$ °C. This can be explained with a delay in boiling in the primary nozzle for low nozzle inlet temperatures. Higher inlet temperatures do not yield a delay in boiling resulting in a significant influence of the compressibility, i.e., $\varepsilon \neq 1$. The new ejector model uses a constant value for the effective flow area A_{eff} from equation (7.6). This constant parameter will have to be replaced by more advanced correlations describing the flow through the primary nozzle based on further measurements.

7.8 Numerical Results

This section presents some numerical results obtained with the new ejector model presented in section 7.6. The two refrigeration systems shown in figure 7.10 were simulated. Both systems were composed of steady-state component models from the `HVAC` package that is part of the new component model library presented in chapter 4. The steady-state models are described in appendix C. Table 7.3 lists the boundary conditions for both steady-state simulations.

Component	**Specified Parameters**
Compressor	$n = 1,000$ 1/min, $V = 30$ cm^3, $\eta_{\mathit{eff},is} = 70\%$, $\lambda_{\mathit{eff}} = 70\%$
Internal Heat Exchanger	$\eta = 70\%$
Evaporator	$p_{12} = 35$ bar

Table 7.3: Parameter settings for COP comparison between conventional CO_2 refrigeration system and CO_2 ejector refrigeration system.

The ejector model uses a simple correlation for the ejector efficiency η_e depending on the mass flow ratio ψ and a constant value for the effective flow area A_{eff} as presented in the previous section.

Simulations were carried out for various high pressures and different gas cooler outlet temperatures. Figure 7.11 shows the results of the simulations. The COP is normalized to the maximum COP of the reference system at a gas cooler outlet temperature $\vartheta_4 = 45$ °C. It can be seen that the total efficiency of a CO_2 refrigeration system can be improved by up to 15% when the prototype ejector designed and built at the IfT is used instead of the conventional throttling valve. Similar results were obtained in other projects for example by Zha et al. (2007).

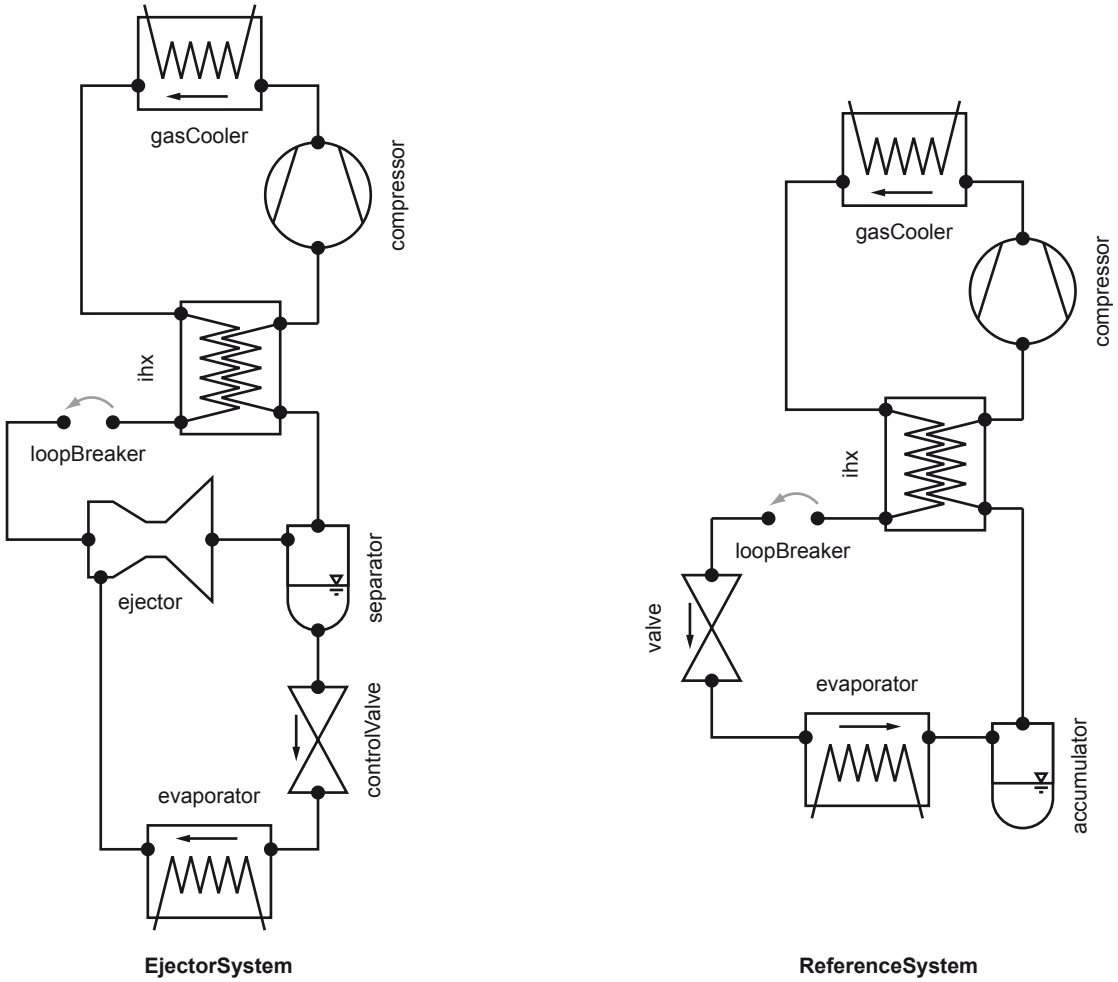


Figure 7.10: `EjectorSystem` and `ReferenceSystem` used for a COP comparison composed of models from the `HVAC` package which is part of the new component model library.

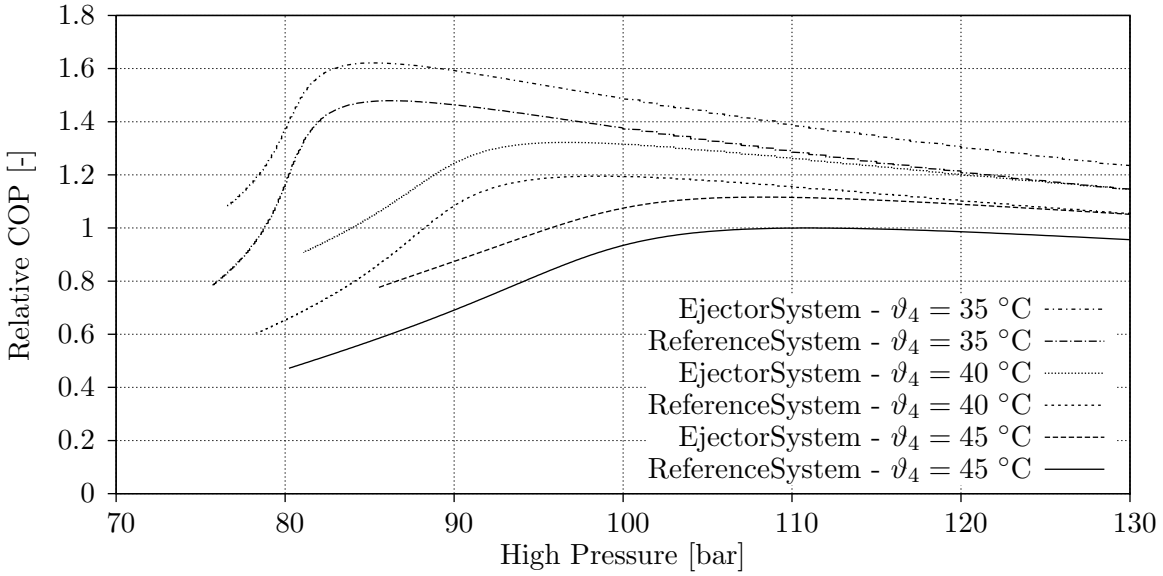


Figure 7.11: Results of high pressure variation for various gas cooler outlet temperatures. The COP is normalized to the maximum COP of the reference system with 45°C gas cooler outlet temperature.

The results obtained from the steady-state simulation of the ejector system using the new ejector model motivate further research in this field. The equations used in the new ejector model can easily be transferred to an ejector model that can be used with other component models from the `HVAC_p` package of the new component model library. This would allow for a more detailed analysis of the described systems including the simulation of transient operation conditions. Providing component models with different levels of detail in a single component model library allows for a basic evaluation of new thermodynamic systems as presented in this chapter as well as for a detailed analysis as presented in the previous chapter and thus supports the entire spectrum of possible users of the library.

Chapter 8

Conclusions and Outlook

The new object-oriented component model library for thermodynamic systems meets the major objective by supporting the entire spectrum of possible users from developers to design engineers. This chapter summarizes all important topics discussed in the previous chapters and outlines the visions for future developments.

8.1 Conclusions

System simulations are an important foundation of modern engineering. They allow for a deeper understanding and for a reliable prediction of the behavior of existing and future systems. The major objective of this thesis was the proposal of new object-oriented model libraries for thermodynamic systems. The proposed libraries should overcome the limitations of existing model libraries in this field by meeting the requirements of the entire spectrum of users from developers to design engineers. Two major design principles were identified that are important to achieve this goal: first of all, the model libraries and the models themselves have to feature an object-oriented structure that is simple to understand and that allows for multiple extensions. In addition, the automated visualization in relevant diagrams has to be supported throughout the entire process of development and application. A fluid property library was presented that allows for a numerically efficient inclusion of external fluid property computation codes in Modelica and in various software tools. This new fluid property library was used in a new component model library for the simulation of thermodynamic systems developed based on the proposed design rules. The capabilities of the new libraries were demonstrated in two example applications covering different thermodynamic systems.

Modelica was selected amongst other equation-based modeling languages due to its strong object-oriented features and its open-source character. The structure of the new component model library was based on an object-oriented analysis and follows a set of novel design rules. Graphical representations for all object-oriented features of the Modelica language based on the UML standard were formally introduced for the first time to allow for a scientific discussion of object-oriented structures in this thesis and beyond. The object-oriented analysis was based on two simple relations: the *is-a*-relation and the *part-of*-relation. The proposed

design rules specify a limited depth of the resulting inheritance tree, forbid complex nestings, and constrict multiple inheritance. They are formulated in a general way and can be applied to the design of any new component model library. In order to support the object-oriented analysis, a new software tool was presented that allows for an automated generation of class diagrams based on the defined graphical representations.

A new generalized approach was presented that allows for the inclusion of external fluid property code in Modelica models using the standard fluid property interface provided in the Modelica Standard Library. Based on an object-oriented analysis and the design rules mentioned above, a new simpler object-based approach for a fluid property library was developed that provides separate models for different fluid types such as gases, liquids, or refrigerants. The models from this new fluid property library were used in all new component models. An external interface library was developed to handle multiple external fluid property computation codes in a numerically efficient way. This interface library can easily be extended to include new external fluid property computation codes. It was shown that the new approach to include fluid properties in Modelica is compatible to state-of-the-art medium models written in Modelica. The new approach supports the automated visualization of numerical results during and after the solution process and provides interfaces to many software tools such as MS Excel of MATLAB/Simulink.

The new component model library provides models for all components required for the steady-state as well as for the transient simulation of refrigeration, air-conditioning, and heat-pump systems. The formulation of conservation laws and of pressure drop and heat transfer correlations including smooth transition functions to switch between different regions were presented in detail. A special approach to handle the time derivative of pressure based on a previous work by Lemke (2005) was consistently introduced throughout the entire component model library. The heat exchanger model was presented in detail to demonstrate the feasibility of the new object-oriented structure for complex component models. The management of system information such as the refrigerant mass or the inner volume was integrated in a single component model that is required in each system. The new component model library supports the automated generation of thermodynamic plots in relevant diagrams such as pressure-enthalpy diagrams. A software tool was developed that allows for the automated generation of thermodynamic phase diagrams during and after simulations. This software tool uses the new fluid property library to create all thermodynamic phase diagrams.

Two demonstrating examples were provided to show the capabilities of the new component model library. The first demonstrating example was a prototype Peltier water-water heat exchanger. The component model library was extended by the model of a Peltier element. Furthermore, electrical components from the Modelica Standard Library were used to set up the system simulation. The numerical results obtained from the transient simulation showed good agreement with the results obtained from measurements. The second demonstrating example was a prototype ejector partly developed within the scope of this thesis. A new efficiency-based model for an ejector was presented and was used to evaluate the potential COP improvements when using an ejector as a replacement for the valve in a CO_2 refrigeration system.

The developed component model library is already used and extended by a team of users and developers at the Institut für Thermodynamik and at the TLK-Thermo GmbH. It has been deployed in the simulation of stationary and mobile heat-pump systems, to evaluate novel system designs for thermodynamic systems, and to develop control strategies. It offers a structure that supports the entire spectrum of users from developers to design engineers.

8.2 Future Development

The development of the new component model library and of the new fluid property library has been an iterative process driven by the major design goals described in chapters 2 to 4 resulting in some minor flaws that have to be removed in a future version. The development was furthermore influenced by the development of the Modelica language itself over the last three years. Especially the new Modelica 3.0 standard released on September 10th, 2007, will require some minor adaptations in both libraries. Besides these very technical aspects there is always room for extensions. This section outlines the vision for future developments and points out possible milestones.

Improved Support for Design Engineers

Three types of users were mentioned in the first section of this thesis: developers, simulation specialists, and design engineers. The proposed component model library does support all three of these user types. Nevertheless, the support for design engineers could be improved significantly in the future by providing a new software tool for system computations and parameter studies. Starting point for the future developments should be the executable generated by any of the available Modelica simulators such as Dymola, MathModelica, or SimulationX. This executable combines system model and solver and can be called after specifying all input parameters to simulate the initial system used for the creation of the executable. The new software tool has to provide a graphical user interface (GUI) that allows for the management of different executables, the specification of relevant parameters, and the management of result files. It should incorporate the visualization tool presented in section 5.3 to generate thermodynamic phase diagrams and other plots. Within the scope of this thesis, a first beta version of such a software tool was implemented as a proof-of-concept.

Models for Real-Time Applications

Models for real-time applications have gained much attention over the last couple of years especially in hardware-in-the-loop simulations and non-linear model predictive control. The new component model library developed within the scope of this thesis could be extended to include component models for real-time applications. The new component models can be partly derived by simplifying existing component models already available in the proposed library. The proposed external interface library for fluid property computations could be extended to include simple and fast medium models suitable for real-time applications. Due to the standardized interface, a direct comparison of simple and advanced medium mod-

els can be used to evaluate their performance versus accuracy. The new models could be used in hardware-in-the-loop simulations or for a non-linear model predictive control in real applications.

The main purpose of such non-linear models for a predictive controller is to recognize the trends of a real plant instead of capturing all phenomenological effects to optimize cost functionals. First efforts to simplify models such as the heat exchanger model were carried out successfully using the provided interfaces. Thus, a number of projects have been proposed in close cooperation with computer and software specialists and companies to speed-up the development of models for real-time applications based on the model libraries developed in this thesis.

Further Numerical Aspects

The *dp/dt*-approach used in all component models in the `HVAC_p` package of the new component model library to reduce the number of pressure states as explained in sections 4.3 and 4.11 results in a DAE with less and/or different numerical states compared to other approaches. A detailed numerical analysis should be carried out to better understand these differences and its consequences from a mathematical point of view. The outcome of this analysis could help to improve the *dp/dt*-approach and to define sharp acceptance levels.

Another important numerical aspect is the non-linear solution process during initialization. Tegethoff (1999) allows for the explicit selection of algebraic variables and residues in his platform thus incorporates the expertise of experienced simulation experts. Unfortunately Modelica does not offer means to manually influence the transformation of equations and the selection of appropriate tearing variables although these issues have been discussed at a number of Modelica design meetings. A new proposal should be formulated to include features to influence the equation transformation into a future version of the Modelica language.

Chapter 9

References

M. Abadi and L. Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, 1996.

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

B. Adiprasito. *Simulation des instationären Verhaltens einer Pkw-Klimaanlage mit CO_2 als Kältemittel*. VDI-Verlag, 1998.

N. Ahlbrink. Optimierung eines objektorientierten Wärmeübertragers in Modelica. Studienarbeit, TU Braunschweig, January 2007.

S. E. Andersen, A. Jakobsen, and B. D. Rasmussen. User Interface Design Considerations EESCoolTools - A Collection of Simulation Models for Reftigeration Applications. In *Proc. of SIMS 99*, Linköping, Sweden, October 1999.

P. J. Ashenden, G. D. Peterson, and D. A. Teegarden. *The System Designer's Guide to VHDL-AMS. Analog, Mixed-Signal, and Mixed-Technology Modeling*. Morgan Kaufmann, 2002.

H. D. Baehr. *Thermodynamik: Grundlagen und technische Anwendungen*. Springer-Verlag, 11th edition, 2002.

H. D. Baehr and K. Stephan. *Wärme- und Stoffübertragung*. Springer, Berlin, 4th edition, 2004.

M. Becker. *Automatisierung kältetechnischer Anlagen auf Basis der mathematischen Modellierung des Gesamtsystems*. Fortschritt-Berichte VDI Reihe 19 Nr. 86, VDI Verlag, Düsseldorf, 1996.

A. Bejan. *Advanced Engineering Thermodynamics*. John Wiley and Sons, 1988.

M. Bockholt, W. Tegethoff, N. Lemke, N.-C. Strupp, and C. Richter. Transient Modelling of a Controllable Low Pressure Accumulator in CO_2 Refrigeration Cycles. In *Proc. of 6th International Modelica Conference*, Bielefeld, March 2008.

W. S. Bodinus. The Rise and Fall of Carbon Dioxide Systems. *ASHRAE Journal*, pages 37–42, April 1999.

G. Booch. *Objektorientierte Analyse und Design. Mit praktischen Anwendungsbeispielen.* Addison-Wesley, 6th edition, 1995.

K. Bos, R. Huebener, and C. Tsuei. Prospects for Peltier cooling of superconducting electronics. *Cryogenics*, 38(3):325–328, March 1998.

E. Bou Lawz Ksayer. *Study and Design of Systems with Improved Energy Efficiency Operating with CO_2 as Refrigerant.* PhD thesis, Centre Energétique et Procédés - Ecole des Mines de Paris, November 2007.

K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* Soc for Industrial & Applied Math, 1996.

J. A. Burke and J. Haws. Vehicle Thermal Systems Modeling Using Flowmaster2. In *Proc. of Vehicle Thermal Management Systems Conference & Exposition*, number 2001-01-1696, Warrendale, PA, May 2001.

F. Casella and A. Leva. Modelling of distributed thermo-hydraulic processes using Modelica. In *Proceeding of the IV MathMod Conference*, pages 514–521, Vienna, September 2003.

F. Casella and A. Leva. Modelling of thermo-hydraulic power generation processes using Modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12(1):19–33, February 2006.

F. Casella, M. Otter, K. Proelß, C. Richter, and H. Tummescheit. The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In *Proc. of 5th International Modelica Conference*, pages 631–640, Vienna, September 2006.

J. C. Chen. Correlation for boiling heat transfer to saturated fluids in convective flow. *I & EC Process Design and Development*, 5(3):322–329, 1966.

E. Christen and K. Bakalar. VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications. *IEEE Transactions on Circuits and Systems -II: Analog and Digital Signal Processing*, 46(10):1263–1272, October 1999.

C. Clauß, A. Schneider, and P. Schwarz. Schaltungssimulation mit Modelica/Dymola. In *Proc. of Diskussionssitzung Entwicklung von Analogschaltungen mit CAE-Methoden (ANALOG)*, pages 177–182, Bremen, June 2002.

B. Cullimore and T. Hendricks. Design and Transient Simulation of Vehicle Air Conditioning Systems. In *Proc. of Vehicle Thermal Management Systems Conference & Exposition*, number 2001-01-1692, Warrendale, PA, May 2001.

O.-J. Dahl and K. Nygaard. SIMULA - an ALGOL-Based Simulation Language. *Communications of the ACM*, 9(9):671–678, September 1966.

DENSO Cooperation. Staying the Course - Annual Report 2003, 2003.

P. A. Domanski. The Use of an Ejector as a Refrigerant Expander. In *Proc. of 1990 USNC/IIR-Purdue Refrigeration Conference*, pages 10–19, Purdue University, July 1990.

M. Drescher, A. Hafner, A. Jakobsen, P. Neksa, and S. Zha. Experimental Investigation of Ejector for R-744 Transcritical Systems. In *Proc. of International Congress of Refrigeration*, Beijing, 2007.

J. Eborn. *On Model Libraries for Thermo-hydraulic Applications*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, March 2001.

B. Eckel. *Thinking in Java*. Prentice Hall PTR, 4th edition, 2006.

S. Elbel and P. Hrnjak. Experimental Validation and Design Study of a Transcritical CO_2 Prototype Ejector System. In *Proc. of 7th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Trondheim, Norway, May 2006a.

S. Elbel and P. Hrnjak. Experiments and Modeling of Ejector Systems. Presentation at c-dig (the carbon dioxide interest group) Meeting, Purdue University, West Lafayette, IN, March 2006b.

H. Elmqvist. *A structured model language for large continuous systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, 1978.

H. Elmqvist and S. E. Mattson. Modelica - The Next Generation Modeling Language, An International Effort. In *Proc. of the 1st World Congress on System Simulation*, Singapore, September 1997.

H. Elmqvist, H. Tummescheit, and M. Otter. Object-Oriented Modeling of Thermo-Fluid Systems. In *Proc. of 3rd International Modelica Conference*, pages 269–286, Linköping, November 2003.

J. Farman, B. Gardiner, and J. Shanklin. Large losses of total ozone in Antarctica reveal seasonal ClO_x/NO_x interaction. *Nature*, 315:207–210, May 1985.

M. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics*. Springer-Verlag, Berlin, 3rd edition, 2002.

M. Fowler. *UML Distilled. Third Edition. A Brief Guide to the Standard Object Modeling Languange*. Addison-Wesley Longman, Amsterdam, 2003.

M. Fowler. *Analysis Patterns: Reusable Object Models*. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, 1996.

P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

S. Försterling. *Vergleichende Untersuchung von CO_2-Verdichtern in Hinblick auf den Einsatz in mobilen Anwendungen*. Doktorarbeit, TU Braunschweig, December 2003.

N. H. Gay. REFRIGERATION SYSTEM. *U.S. Patent 1,836,318*, December 1931.

J. Haase, E. Hessel, A. Graßmann, and J. Schäfer. Model Exchange Process in Automotive Industry with VHDL-AMS – Philosophy and Examples. In *Tagungsband, 2. Aachen Electronics Symposium 2004*, September 2004.

E. Hairer and G. Wanner. *Vol.2 : Stiff and Differential-Algebraic Problems*. Springer, Berlin, 1996.

H. Hasse, M. Becker, K. Grossmann, and G. Maurer. Top-Down Model for Dynamic Simulation of Cold Storage Plants. *International Journal of Refrigeration*, 19(1):10–18, 1996.

C. Heinrich and K. Berthold. A Modelica Library for Simulation of Household Refrigeration Appliances, Features and Experiments. In *Proc. of 5th International Modelica Conference*, pages 677–684, Vienna, September 2006.

C. Junior. Energetische Gegenüberstellung von Kältesystemvarianten mit dem Arbeitstoff CO_2 für den Einsatz in der Tiefkühltechnik. Diplomarbeit, TU Braunschweig, February 2006.

J. Köhler. Skriptum zur Vorlesung: Thermodynamik. Institut für Thermodynamik, TU Braunschweig, 2007a.

J. Köhler. Skriptum zur Vorlesung: Wärme- und Stoffübertragung. Institut für Thermodynamik, TU Braunschweig, 2007b.

J. Köhler, W. Tegethoff, C. Richter, and C. Tischendorf. Experimental and theoretical study of a CO_2 ejector refrigeration cycle. In *VDA Alternative Refrigerant Winter Meeting*, Saalfelden, 2007.

S. A. Klein. *EES - Engineering Equation Solver - User Manual*. F-Chart Software, Middleton, WI, 1999.

A. Kornhauser. The Use of an Ejector as a Refrigerant Expander. In *Proc. of USNC/IIR Purdue Refrigeration Conference*, pages 10–19, Purdue, Indiana, 1990.

R. Kossel. Objektorientierte Modellierung von Wärmeübertragern in Modelica. Studienarbeit, TU Braunschweig, June 2005.

R. Kossel, W. Tegethoff, M. Bodmann, and N. Lemke. Simulation of complex systems using Modelica and tool coupling. In *Proc. of 5th International Modelica Conference*, pages 485–490, Vienna, September 2006.

R. Krauss, J. Luettmer-Strathmann, J. Sengers, and K. Stephan. Transport Properties of 1,1,1,2-Tetrafluoroethane (R134a). *International Journal of Thermophysics*, 14(4):951–987, 1993.

N. C. Lemke. *Untersuchung zweistufiger Flüssigkeitskühler mit dem Kältemittel CO_2*. Forschungsberichte des Deutschen Kälte- und Klimatechnischen Vereins Nr. 73, DKV, Stuttgart, 2005.

E. W. Lemmon, M. Huber, and M. McLinden. NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 8.0. National Institute of Standards and Technology, Standard Reference Data Program, Gaithersburg, 2007.

D. Li and E. A. Groll. Transcritical CO_2 refrigeration cycle with ejector-expansion device. *International Journal of Refrigeration*, 28(5):766–773, 2005.

D. Li and E. A. Groll. Analysis of an Ejector Expansion Device in a Transcritical CO_2 Air Conditioning System. In *Proc. of 7th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Trondheim, Norway, May 2006.

D. Limperich, M. Braun, G. Schmitz, and K. Prölß. System Simulation of Automotive Refrigeration Cycles. In *Proc. of the 4th International Modelica Conference*, pages 193–199, Hamburg-Harburg, March 2005.

M. Loeffler. Ein Werkzeug zur Strukturanalyse von MODELICA-Bibliotheken. Diplomarbeit, TU Braunschweig, October 2006.

M. Loeffler, M. Huhn, C. Richter, and R. Kossel. ModelicaCDV - A Tool for Visualizing the Structure of Modelica Libraries. In *Proc. of 5th International Modelica Conference*, pages 55–62, Vienna, September 2006.

G. Lorentzen. Throttling - The Internal Hemorrhage of the Refrigeration Process. In *Proc. of Institute of Refrigeration*, volume 80, pages 39–47, 1983.

G. Lorentzen. TRANS-CRITICAL VAPOUR COMPRESSION CYCLE DEVICE. *EP 1989910211*, issued October 1993. filed September 1989.

J. Martin and J. J. Odell. *Object-Oriented Methods: A Foundation, UML Edition.* Pearson Education Ltd., 2nd edition, 1998.

K. Matsuo, K. Sasaguchi, Y. Kiyotoki, and H. Mochizuki. Investigation of Supersonic Air Ejectors : Part 2, Effects of Throat-Area-Ratio on Ejector Performance. *Bulletin of JSME*, 25(210):1898–1905, December 1982. ISSN 00213764.

S. Mattsson and M. Andersson. The Ideas Behind Omola. In *Proc. of 1992 IEEE Symposium on Computer Aided Control System Design*, pages 23–29, Napa, CA, March 1992.

F. Mayinger. *Strömung und Wärmeübergang in Gas-Flüssigkeits-Gemischen.* Springer-Verlag, Berlin, 1982.

J. Mazen. Auswirkung der partiellen Einschränkung der Kondensatorluftzufuhr bei Pkw-Klimaanlagen mit den Kältemitteln R134a und R744. Diplomarbeit, TU Braunschweig, May 2007.

M. McLinden, E. Lemmon, and R. Jacobsen. Thermodynamic properties for the alternative refrigerants. *International Journal of Refrigeration*, 21(4):332–338, 1998.

J. R. McNeill and P. Kennedy. *Something New Under the Sun: An Environmental History of the Twentieth-Century World*. Global Century Series. W. W. Norton & Company, 2001.

P. Menegay. *A Computational Model for Two-Phase Ejector Flow*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, January 1997.

S. Meyers. *Effective C++: 50 Specific Ways to Improve Your Programs and Design*. Addison-Wesley Professional, 2nd edition, 1997.

Modelica Association. *Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 2.2*, February 2nd, 2005.

Modelica Association. *Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.0*, September 5th, 2007.

R. E. Nance. Simulation Programming Languages: An Abridged History. In *Proc. of the 1995 Winter Simulation Conference*, pages 1307–1313, Arlington, VA, December 1995.

OMG. The Object Management Group (OMG), 2007. URL `www.omg.org`.

T. M. Ortiz, D. Li, and E. A. Groll. Evaluation of the Performance Potential of CO_2 as a Refrigerant in Air-To-Air Air Conditioners and Heat Pumps: System Modeling and Analysis. Technical Report 1275-2, Final Report of ARTI Project 610-10030, Herrick Labs 2003-20, 2003.

M. Otter, B. Bachman, H. Elmqvist, S. Mattson, C. Schlegel, P. Beater, H. Tummescheit, C. Clauß, A. Schneider, P. Schwarz, H. Wiesman, and M. Tiller. Objektorientierte Modellierung Physikalischer Systeme, Teil 1 - 17. *at Automatisierungstechnik*, 1999-2000.

Y. Ozaki, H. Takeuchi, and T. Hirata. Regeneration of Expansion Energy by Ejector in CO_2 Cycle. In *Proc. of 6th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Glasgow UK, 2004.

S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Tylor & Francis, 1980.

T. Pfafferott. *Dynamische Simulation von CO_2-Kälteprozessen für mobile Kälteanwendungen*. Doktorarbeit, Technische Universität Hamburg-Harburg, 2004.

B. C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

W. Polifke and J. Kopitz. *Wärmeübertragung, Grundlagen, analytische und numerische Methoden*. Pearson Studium, Munich, 2005.

A. Premoli, D. Francesco, and A. Prima. An empirical correlation for evaluating two-phase mixture density under adiabatic conditions. In *European Two-Phase Flow Group Meeting*, Milan, Italy, 1970.

A. A. B. Pritsker and N. R. Hurst. GASP IV: A combined continuous - discrete FORTRAN-based simulation language. *SIMULATION*, 21(3):65–70, 1973.

H. Raiser. *Untersuchung des transienten Verhaltens von CO_2-PKW-Klimaanlagen mit Niederdrucksammler*. Doktorarbeit, TU Braunschweig, January 2005.

C. Richter and F. Casella. ExternalMedia: a Library for Easy Re-Use of External Fluid Property Code in Modelica. In *Proc. of 6th International Modelica Conference*, Bielefeld, March 2008.

C. Richter, C. Tischendorf, R. Fiorenzano, P. Cavalcante, W. Tegethoff, and J. Köhler. Using Modelica as a Design for an Ejector Test Bench. In *Proc. of 5th International Modelica Conference*, pages 501–508, Vienna, September 2006.

D. M. Robinson and E. A. Groll. Theoretical Performance Comparison of CO_2 Transcritical Cycle Technology Versus HCFC-22 Technology for a Military Packaged Air Conditioner Application. *International Journal HVAC&R Research*, 6(4):325–348, October 2000.

A. Roccatello, S. Mancò, and N. Nervegna. Modelling a Variable Displacement Axial Piston Pump in a Multibody Simulation Environment. *Journal of Dynamic Systems, Measurement, and Control*, 129(4):456–468, July 2007.

D. Rowe, editor. *Thermoelectrics Handbook, Macro to Nano*. Taylor & Francis, 2006.

F. Schiavo and F. Casella. Object-Oriented Modelling and Simulation of Heat Exchangers with Finite Element Methods. *Mathematical and Computer Modelling of Dynamic Systems*, 13(3):211–235, June 2007.

R. Schmidt. *Eine Methode zur numerischen Untersuchung von Strömung und Wärmeübertragung in komplexen Geometrien*. Doktorarbeit, TU Braunschweig, Juni 2002.

M. Shah. A new correlation for heat transfer during boiling flow through pipes. *ASHRAE Transactions*, 82(1):66–86, 1976.

M. Shah. A general correlation for heat transfer during film condensation inside pipes. *International Journal of Heat and Mass Transfer*, 22(4):547–556, April 1979.

M. Siebenhaller. Automatisches Layout von UML-Klassendiagrammen. Diplomarbeit, Eberhard-Karls-Universität Tübingen, June 2003.

R. Span. *Multiparameter Equations of State, An Accurate Source of Thermodynamic Property Data*. Springer, 2000. ISBN 3-540-67311-3.

R. Span and W. Wagner. A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa. *Journal of Physical and Chemical Reference Data*, 25(6):1509–1596, 1996.

K. Stephan. *Wärmeübergang beim Kondensieren und beim Sieden*. Springer-Verlag, Berlin, 1988.

N.-C. Strupp, N. Lemke, W. Tegethoff, and J. Köhler. Investigation of Low Pressure Accumulators in CO_2 Refrigeration Cycles. In *Proc. of International Congress on Refrigeration*, Beijing, China, 2007. ICR07-E1-1480.

Sun Microsystems, Inc. Swing (Java^{TM} Foundation Classes). URL `http://java.sun.com/javase/6/docs/technotes/guides/swing/`.

W. Tegethoff. *Eine objektorientierte Simulationsplattform für Kälte-, Klima- und Wärmepumpensysteme*. Fortschritt-Berichte VDI Reihe 19 Nr. 118, VDI Verlag, Düsseldorf, 1999.

W. Tegethoff, N. Lemke, and J. Köhler. Component modelling and specification using a new approach for transient simulation. In *VDA Alternative Refrigerant Winter Meeting - Automotive Air-Conditioning and Heat Pump Systems*, Saalfelden, 2004.

W. Tegethoff, T. Horst, C. Richter, R. Kossel, M. Bodmann, and N. Lemke. Modeling and Simulation of Compressors in Mobile Air-Conditioning Systems. In *Proc. of EUROSIM 2007*, Ljubljana, Slovenia, September 2007.

M. M. Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, 2001. ISBN 0-7923-7367-7.

R. Tillner-Roth, J. Li, A. Yokozeki, H. Sato, and K. Watanabe. Thermodynamic Properties of Pure and Blended Hydrofluorocarbon (HFC) Refrigerants. *Japan Society of Refrigerating and Air-Conditioning Engineers*, 1998.

TLK-Thermo GmbH. *StateViewer - Documentation, Version 1.3*. Braunschweig, November 2007.

H. Tummescheit. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, August 2002.

H. Tummescheit and J. Eborn. Chemical Reaction Modeling with ThermoFluid/MF and MultiFlash. In *Proc. of 2nd International Modelica Conference*, pages 31–39, Oberpfaffenhofen, March 2002.

H. Tummescheit, J. Eborn, and F. J. Wagner. Development of a Modelica Base Library for Modeling of Thermo-Hydraulic Systems. In *Proc. of Modelica Workshop 2000*, pages 41–51, Lund, October 2000.

H. Tummescheit, J. Eborn, and K. Prölß. AirConditioning - a Modelica Library for Dynamic Simulation of AC Systems. In *Proc. of 4th International Modelica Conference*, pages 185–192, Hamburg-Harburg, March 2005a.

H. Tummescheit, J. Eborn, K. Prölß, S. Försterling, and W. Tegethoff. *PKW-Klimatisierung IV, Klimakonzepte, Zuheizkonzepte, Regelungsstrategien und Entwicklungsmethoden*, volume 57, chapter Air-Conditioning: Eine Modelica-Bibliothek zur dynamischen Simulation von Kältekreisläufen, pages 196–214. expert verlag, 2005b.

VDI, editor. *VDI-Wärmeatlas: Berechnungsblätter für den Wärmeübergang*. Springer-Verlag, 9th edition, 2002.

H. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Pearson Education Ltd., 1995.

V. Vesovic, W. Wakeham, G. Olchowy, J. Sengers, J. Watson, and J. Millat. The Transport Properties of Carbon Dioxide. *Journal of Physical and Chemical Reference Data*, 19(3): 763–808, 1990.

L. Viklund and P. Fritzson. ObjectMath - an object-oriented language and environment for symbolic and numerical processing in scientific computing. *Scientific Programming*, 4(4): 229–250, Winter 1995.

S. von Haaf. Wärmeübergang in Luftkühlern. In F. Steimle and K. Stephan, editors, *Wärmetauscher*. Springer-Verlag, Berlin, 1988.

W. Wagner. *Strömung und Druckverlust*. Kamprath-Reihe. Vogel Fachbuch, 5th edition, 2001.

W. Wagner and U. Overhoff. *ThermoFluids*. Springer-Verlag Berlin, 2006.

M. Wein. *Numerische Simulation von kritischen und nahkritischen Zweiphasenströmungen mit thermischen und fluiddynamischen Nichtgleichgewichtseffekten*. Doktorarbeit, Technische Universität Dresden, April 2002.

J. Winkler, V. Aute, B. Yang, and R. Radermacher. Potential benefits of thermoelectric elements used with air-cooled heat exchangers. In *Proc. of 2006 International Refrigeration and Air Conditioning Conference at Purdue*, volume 1, pages R091.1–R091.8, West Lafayette, July 2006.

M. Woldesemayat and A. Ghajar. Comparison of void fraction correlations for different flow patterns in horizontal and upward inclined pipes. *International Journal of Multiphase Flow*, 33(4):347–370, April 2007.

S. Zha, A. Jakobsen, A. Hafner, and P. Neksa. Design and Parametric Investigation on Ejector for R-744 Transcritical System. In *Proc. of International Congress of Refrigeration 2007*, number ICR07-B1-743, Beijing, 2007.

K. Zorbas, E. Hatzikraniotis, and K. Paraskevopoulos. Power and Efficiency Calculation in Commercial TEG and Application in Wasted Heat Recovery in Automobile. In *Proc. of 5th European Conference on Thermoelectrics*, 2007.

Appendix A

Nomenclature

Latin Characters

A	area $[m^2]$
A	cross sectional area $[m^2]$
A_a	total outer surface area $[m^2]$
A_{eff}	effective flow area $[m^2]$
A_0	smallest flow area $[m^2]$
c_p	specific heat capacity at constant pressure $[\frac{J}{kg \cdot K}]$
d	density $[\frac{kg}{m^3}]$
d	diameter $[m]$
d	thickness $[m]$
d_a	outer tube diameter $[m]$
d_h	hydraulic diameter $[m]$
E	total energy $[J]$
F	factor in eqn. (4.41) [-]
Fr	Froude number [-]
g	gravitational constant $[\frac{m^2}{s}]$
h	specific enthalpy $[\frac{J}{kg}]$
h_{1+x}	specific enthalpy of moist air per kilogram of dry air $[\frac{J}{kg_{\text{dry air}}}]$
H	enthalpy $[J]$
I	momentum $[\frac{kg \cdot m}{s}]$
I	current $[A]$
$\dot{I}$	momentum flow $[\frac{kg \cdot m}{s^2}]$
k	heat transfer coefficient $[\frac{W}{m^2 \cdot K}]$
l_{lam}	length of finned tube $[m]$
L	length $[m]$
m	mass $[kg]$
$\dot{m}$	mass flow rate $[\frac{kg}{s}]$
$\dot{m}_D$	driving mass flow rate $[\frac{kg}{s}]$
$\dot{m}_S$	suction mass flow rate $[\frac{kg}{s}]$
M	total mass in control volume $[kg]$
M	torque $[N \cdot m]$
MM	molar mass $[\frac{kg}{mol}]$
n	exponent [-]
n	speed $[\frac{1}{s}]$
nX	number of mass fractions [-]
nXi	number of independent mass fractions [-]
Nu	Nusselt number [-]
p	pressure $[Pa]$
p_{evap}	evaporation pressure level $[Pa]$
p_h	high pressure level $[Pa]$
p_{im}	intermediate pressure level $[Pa]$
Δp	pressure loss $[Pa]$
Δp_f	friction pressure loss $[Pa]$
P	power $[W]$
P_{el}	electrical power $[W]$
Pr	Prandtl number [-]
$\dot{q}$	heat flux density $[\frac{W}{m^2}]$
$\dot{Q}$	heat flow rate $[W]$
$\dot{Q}_C$	absorbed heat flow rate $[W]$
$\dot{Q}_H$	rejected heat flow rate $[W]$
r	radius $[m]$
R	gas constant $[\frac{J}{kg \cdot K}]$
R	two-phase multiplier in eqn. 4.49 [-]
R	thermal resistance $[\frac{K}{W}]$

R	electrical resistance [Ω]
Re	Reynolds number [-]
s	specific entropy [$\frac{J}{kg \cdot K}$]
s	slip factor [-]
s_{1+x}	specific entropy of moist air per kilogram of dry air [$\frac{J}{kg_{\text{dry air}} \cdot K}$]
s_l	distance between serial tubes [m]
s_q	distance between parallel tubes [m]
S	factor in eqn. (4.41) [-]
t	time [s]
t_R	distance between fins [m]
T	temperature [K]
T_w	wall temperature [K]
T_C	temperature at cold side [K]
T_H	temperature at hot side [K]
T_M	average temperature [K]
u	specific internal energy [$\frac{J}{kg}$]
U	internal energy [J]
U	perimeter [m]
v	specific volume [$\frac{m^3}{kg}$]
v_l	air velocity in clear cross section [$\frac{m}{s}$]
V	volume [m^3]
V	displacement volume [m^3]
V_{kv}	control volume [m^3]
w	velocity [$\frac{m}{s}$]
$w_{l,m}$	average air velocity [$\frac{m}{s}$]
w_m	average velocity [$\frac{m}{s}$]
We	Weber number [-]
$\dot{W}_t$	shaft work (*technische Arbeit*) [W]
x	quality [-]
x	state variable [-]
x^*	flow steam quality [-]
x_t	transition point
x_w	water content per kilogram of dry air [$\frac{kg}{kg_{\text{dry air}}}$]
Δx	transition length
X	mass fraction [-]
Xi	independent mass fraction [-]
X_{tt}	Martinelli parameter [-]
y	algebraic variable [-]
z	z-coordinate [m]
z_l	number of serial tubes [-]
z_q	number of parallel tubes [-]
Z	figure-of-merit [$\frac{1}{K}$]

Greek Characters

α	coefficient of heat transfer [$\frac{W}{m^2 \cdot K}$]
α	Seebeck coefficient [$\frac{V}{K}$]
α	flow coefficient [-]
α_{ab}	relative Seebeck coefficient [$\frac{W}{A \cdot K}$]
β	isobaric expansion coefficient [$\frac{1}{K}$]
δ	dimensionless density [-]
δ_R	fin thickness [m]
ε	void fraction [-]
ε	expansion coefficient [-]
ζ	friction factor [-]
η	efficiency [-]
η_c	isentropic compressor efficiency [-]
η_e	isentropic ejector efficiency [-]
η_t	isentropic turbine efficiency [-]
κ	isothermal compressibility [$\frac{1}{Pa}$]
κ	thermal conductance [$\frac{W}{m \cdot K}$]
λ	thermal conductivity [$\frac{W}{m \cdot K}$]
λ	pipe friction factor [-]
λ	volumetric efficiency [-]
μ	dynamic viscosity [$Pa \cdot s$]
ν	kinematic viscosity [$\frac{m^2}{s}$]
ξ	steam concentration [-]
ξ	friction factor [-]
π	pressure ratio [$\frac{Pa}{Pa}$]
π_{max}	maximum pressure ratio [$\frac{Pa}{Pa}$]
ρ	electrical resistivity [$\Omega \cdot m$]
ϱ	density [$\frac{kg}{m^3}$]
σ	surface tension [$\frac{N}{m}$]
σ	electrical conductivity [$\frac{S}{m}$]
τ	dimensionless inverse temperature [-]
τ_0	wall shear stress [Pa]
ϕ	dimensionless Helmholtz energy [-]

ϕ	relative humidity [%]	ψ	mass flow ratio [-]
ϕ	coefficient of performance [-]	Ψ	void fraction [-]
φ	phase [*rad*]		

Subscripts

0	initial conditions	*is*	isentropic conditions
ambient	ambient conditions	*l*	liquid properties
c	critical properties	*N*	North
d	discharge properties	*out*	leaving
eff	effective values	*s*	suction properties
E	East	*S*	South
g	gas properties	*W*	West
id	ideal conditions		
in	entering		

Abbreviations

BLT	Block-Lower-Triangle
CFC	Chlorofluorocarbon
COP	Coefficient of Performance
CPU	Central Processing Unit
DAE	Differential Algebraic Equation
EES	Engineering Equation Solver
EOS	Equation of State
GUI	Graphical User Interface
GWP	Global Warming Potential
HCFC	Hydrochlorofluorocarbon
HDL	Hardware Description Language
HFC	Hydrofluorocarbons
HVAC	Heating, Ventilation, and Air-Conditioning
IfT	Institut für Thermodynamik
MBWR	Modified Benedict-Webb-Rubin Equation of State
ModelicaCDV	Modelica Class Diagram Visualizer
NIST	National Institute of Standards and Technology
ODE	Ordinary Differential Equation
PFC	Perfluorocarbon
TIL	TLK-IfT-Library
TILFluids	TLK-IfT-Fluid Property-Library
TISC	TLK Inter Software Connector
UML	Unified Modeling Language

Appendix B

Design Rules

This chapter summarizes all design rules for a good design of object-oriented component model libraries that were developed in chapter 2 based on general considerations about library structuring and experiences from available component model libraries.

Design Rule 1:	Class names should begin with an uppercase letter whereas object names should begin with a lowercase letter. Exceptions to this rule are allowed for object names if they refer to common abbreviations marked by an uppercase letter and if chances are low to mistake them as class names.

Design Rule 2:	Inheritance should only be used if the relation between two classes can be described as an *is-a*-relation.

Design Rule 3:	Multiple inheritance should only be used if each inheritance relation fulfills design rule 2.

Design Rule 4:	Multiple inheritance should be avoided whenever possible.

Design Rule 5:	The object-oriented structure of the component model library should be constantly monitored and re-evaluated during the initial design process since it can hardly be changed later on.

Design Rule 6:	Component model libraries should feature an inheritance structure that is as flat as possible.

Appendix C

Steady-State Object-Oriented Modeling of Fluid Systems

This chapter describes most of the steady-state component models in the Modelica library `HVAC` which is part of the new component model library presented in chapter 4 whose structure is shown in figure 4.2. The presented models were developed in cooperation with Tegethoff and Kossel and are used in the Modelica lectures at TU Braunschweig and in the Modelica training courses offered by TLK-Thermo. A brief introduction to the library and to a possible application is also given in Richter et al. (2006).

C.1 Connectors

The connector models used in the `HVAC` library do not support flow reversal which is the main difference to the connector models presented in section 4.2. The two connector models called `FluidConnectorA` and `FluidConnectorB` contain the pressure p, the specific enthalpy h, and the mass flow rate $\dot{m}$ as shown in code listing C.1.

```
import SI = Modelica.SIunits;

connector FluidConnectorA "Fluid inlet connector"
  SI.AbsolutePressure p "Pressure";
  flow SI.MassFlowRate m_flow "Mass flow rate";

  input SI.SpecificEnthalpy h "Specific enthalpy";
end FluidConnectorA;

connector FluidConnectorB "Fluid outlet connector"
  SI.AbsolutePressure p "Pressure";
  flow SI.MassFlowRate m_flow "Mass flow rate";

  output SI.SpecificEnthalpy h "Specific enthalpy";
```

```
end FluidConnectorB;
```

Code Listing C.1: Fluid connector models in `HVAC` library.

The main difference between the two connector models besides the different icons is the use of `input/output` for the specific enthalpy. This ensures that it is not possible to connect two connectors of type `FluidConnectorB` with each other.

C.2 Compressor Model

Two different compressor models are available in the `HVAC` library, a simple and an advanced one. The simple compressor model assumes constant parameter values for the volumetric efficiency λ_{eff} and for the isentropic efficiency $\eta_{eff,is}$. The following set of equations is formulated in the `SimpleCompressor` model

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{C.1a}$$

$$\eta_{eff,is} = \frac{h_{d,is} - h_s}{h_d - h_s} \tag{C.1b}$$

$$\dot{m}_{in} = \varrho_{in}\, n\, V\, \lambda_{eff} \tag{C.1c}$$

where V is the displacement volume and n the speed. The subscript d refers to the discharge side, s to the suction side, and is to isentropic conditions. The advanced compressor model formulates the efficiencies depending on the pressure ratio π, the maximum pressure ratio π_{max} at which the suction mass flow becomes zero, and the speed n. The following relations are for example used in the `AdvancedBock110` model

$$\pi = \frac{p_d}{p_s} \tag{C.2a}$$

$$\lambda_{eff} = \left(\frac{\pi_{max} - \pi}{\pi_{max} - 1}\right)^2 \left(a\, n^2 + b\, n + 1\right) \tag{C.2b}$$

$$\eta_{eff,is} = \frac{\pi_{max} - \pi}{\pi_{max}} + \left(\frac{\pi_{max} - \pi}{\pi_{max}}\right)^{\pi} \tag{C.2c}$$

C.3 Gas Cooler Model

The gas cooler model in the `HVAC` library uses an efficiency η to determine the outlet specific enthalpy. The pressure drop is neglected. The outlet temperature reaches the temperature of the secondary fluid (i.e., the ambient temperature) for the ideal case $\eta = 1$. The following set of equations describes the gas cooler

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{C.3a}$$

$$p_{out} = p_{in} \tag{C.3b}$$

$$\eta = \frac{h_{in} - h_{out}}{h_{in} - h_{out,id}} \tag{C.3c}$$

$$h_{out,id} = h\_pT(p_{out}, T_{ambient}) \tag{C.3d}$$

where $h_{out,id}$ is the ideal outlet specific enthalpy determined using the outlet pressure and the ambient temperature according to equation (C.3d).

C.4 Evaporator Model

The evaporator model in the `HVAC` library uses the analytical exponential solution for the air temperature as derived for example by Tegethoff (1999, equation (134)). Again, the pressure drop is neglected. The specific heat capacity of air $c_{p,air}$ is furthermore assumed to be constant. The inlet air temperature $T_{in,air}$, the air mass flow rate $\dot{m}_{air}$, and the product of heat transfer coefficient k and heat transfer area A are parameters in the evaporator model. The following set of equations is used

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{C.4a}$$

$$p_{out} = p_{in} \tag{C.4b}$$

$$T_{out,air} = T_{in} + (T_{in,air} - T_{in}) \exp\left(\frac{-kA}{c_{p,air}\ \dot{m}_{air}}\right) \tag{C.4c}$$

$$(T_{in,air} - T_{out,air}) c_{p,air}\ \dot{m}_{air} = \dot{m}_{in}(h_{out} - h_{in}) \tag{C.4d}$$

C.5 Internal Heat Exchanger

Different models for internal heat exchangers exist in the `HVAC` library. The one presented in this section can only be used for one-phase fluids on both sides and assumes constant specific heat capacities. An efficiency ϵ is used that describes the amount of heat transferred between the two fluid streams. The following set of equations is used

$$\dot{m}_{1/2,in} + \dot{m}_{1/2,out} = 0 \tag{C.5a}$$

$$p_{1/2} = p_{1/2} \tag{C.5b}$$

$$\dot{m}_{1,in}\ h_{1,in} + \dot{m}_{1,out}\ h_{1,out} + \dot{Q} = 0 \tag{C.5c}$$

$$\dot{m}_{2,in}\ h_{2,in} + \dot{m}_{2,out}\ h_{2,out} - \dot{Q} = 0 \tag{C.5d}$$

$$\epsilon = \frac{T_{1,in} - T_{1,out}}{T_{1,in} - T_{2,out}} \tag{C.5e}$$

where the two subscripts 1 and 2 refer to the high-pressure and the low-pressure side of the internal heat exchanger respectively.

C.6 Accumulator Model

The accumulator model in the `HVAC` library describes a point on the dew line. The following set of equations is used

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{C.6a}$$

$$p_{out} = p_{in} \tag{C.6b}$$

$$h_{out} = h_{in} \tag{C.6c}$$

$$h_{out} = h_g \tag{C.6d}$$

C.7 Valve Model

The valve model in the `HVAC` library is very similar to the valve model used as a demonstrator in chapter 3 and presented in code listing 3.1. The following equations are used

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{C.7a}$$

$$h_{out} = h_{in} \tag{C.7b}$$

$$\dot{m}_{in} = A_{\mathit{eff}} \sqrt{2\ \varrho_{in}(p_{in} - p_{out})} \tag{C.7c}$$

where A_{eff} is the effective flow area.

C.8 Loop Breaker Model

Connecting a couple of the presented component to a closed-loop system results in an overdetermined system of equations since a static mass balance is formulated in each component model. This can be overcome by removing the mass balance equation from one of the component models presented in the previous sections with the disadvantage that the chosen model must exactly be used once in each system. A better approach is to introduce a new component model that does not formulate a mass balance and that can be used as a loop breaker for a closed-loop system. The following set of equations is formulated in this additional component model

$$h_{out} = h_{in} \tag{C.8a}$$

$$p_{out} = p_{in} \tag{C.8b}$$

Appendix D

Partial Derivatives of Fluid Properties

Using external fluid property libraries has the disadvantage that the simulation tool (e.g., Dymola) cannot explicitly compute time derivatives when differentiating property functions during index reduction. A simple solution to this problem is to provide derivative functions for each external function that is differentiated. The derivative functions are implemented in TILFluids using Bridgman's table (see Bejan, 1988) for the one-phase region and a homogeneous model to describe the two-phase region. In the following two sections, all partial derivatives required to compute the time derivative of density are derived for $\varrho(p, h)$. In the last section, all other important partial derivatives with respect to p and h for the one- and the two-phase region are listed.

The time derivative of density can be written as

$$\frac{d\varrho}{dt} = \left(\frac{\partial \varrho}{\partial p}\right)_h \frac{dp}{dt} + \left(\frac{\partial \varrho}{\partial h}\right)_p \frac{dh}{dt} \tag{D.1}$$

In order to compute the two partial derivatives, a case differentiation between a one-phase state and a two-phase state is required. The equations for the two different cases are derived in the following sections. Some equations are simpler to derive for the specific volume instead of the density. The resulting partial derivatives of specific volume can be transformed into partial derivatives of density using the following simple relation

$$\frac{\partial v}{\partial \varrho} = -\frac{1}{\varrho^2} \tag{D.2}$$

D.1 One-Phase State

Bridgman's table (see Bejan, 1988) can be used to derive the equations for the partial derivatives for the one-phase state as a function of β, κ, and c_p yielding

$$\left(\frac{\partial \varrho}{\partial h}\right)_p = -\frac{\beta \varrho}{c_p} \tag{D.3}$$

$$\left(\frac{\partial \varrho}{\partial p}\right)_h = \frac{-T\beta^2 + \beta + \kappa \varrho c_p}{c_p} \tag{D.4}$$

where β is the isobaric coefficient of expansion, κ is the isothermal compressibility, and c_p is the specific heat capacity at constant pressure which can be determined from medium properties

$$\beta = -\frac{1}{\varrho}\left(\frac{\partial \varrho}{\partial T}\right)_p, \quad \kappa = \frac{1}{\varrho}\left(\frac{\partial \varrho}{\partial p}\right)_T, \quad c_p = \left(\frac{\partial u}{\partial T}\right)_p \tag{D.5}$$

D.2 Two-Phase State

The equations for the partial derivatives for the two-phase state can be derived using a homogeneous model to describe the fluid flow

$$v = v_l + \frac{h - h_l}{h_g - h_l}(v_g - v_l) \tag{D.6}$$

where the quotient at the right-hand side describes the quality x of the two-phase refrigerant

$$x = \frac{h - h_l}{h_g - h_l} \tag{D.7}$$

This yields the following differentiation

$$\left(\frac{\partial v}{\partial h}\right)_p = \frac{v_g - v_l}{h_g - h_l} \tag{D.8}$$

The partial derivative of specific volume with respect to pressure can be written as

$$\left(\frac{\partial v}{\partial p}\right)_h = \frac{dv_l}{dp} + \frac{dx}{dp}(v_g - v_l) + x\left(\frac{dv_g}{dp} - \frac{dv_l}{dp}\right) \tag{D.9}$$

where

$$\frac{dx}{dp} = \frac{-\frac{dh_l}{dp}(h_g - h_l) - (h - h_l)\left(\frac{dh_g}{dp} - \frac{dh_l}{dp}\right)}{(h_g - h_l)^2} \tag{D.10}$$

with

$$\frac{dh_l}{dp} = v_l(1 - \beta_l T) + c_{p,l}\frac{dT}{dp} \tag{D.11}$$

$$\frac{dh_g}{dp} = v_g(1 - \beta_g T) + c_{p,g}\frac{dT}{dp} \tag{D.12}$$

The derivatives of density at the phase boundaries can be expressed as

$$\frac{dv_l}{dp} = \beta_l v_l \frac{dT}{dp} - \kappa_l v_l \tag{D.13}$$

$$\frac{dv_g}{dp} = \beta_g v_g \frac{dT}{dp} - \kappa_g v_g \tag{D.14}$$

The derivative of temperature in equation (D.11) - (D.14) can be derived from the Clausius-Clapeyron relation (see Baehr, 2002)

$$\frac{dT}{dp} = T\frac{v_g - v_l}{h_g - h_l} \tag{D.15}$$

D.3 Further Partial Derivatives

This section lists all in TILFluids implemented partial derivatives for the one- and the two-phase region except for the partial derivatives of density which have been derived in details in der last two sections.

Temperature

For the one-phase region

$$\left(\frac{\partial T}{\partial h}\right)_p = \frac{1}{c_p} \tag{D.16}$$

$$\left(\frac{\partial T}{\partial p}\right)_h = \frac{-v + \beta T v}{c_p} \tag{D.17}$$

For the two-phase region

$$\left(\frac{\partial T}{\partial h}\right)_p = 0 \tag{D.18}$$

$$\left(\frac{\partial T}{\partial p}\right)_h = \frac{dT}{dp} \tag{D.19}$$

Specific Entropy

For the one-phase region

$$\left(\frac{\partial s}{\partial h}\right)_p = \frac{1}{T} \tag{D.20}$$

$$\left(\frac{\partial s}{\partial p}\right)_h = -\frac{v}{T} \tag{D.21}$$

For the two-phase region

$$\left(\frac{\partial s}{\partial h}\right)_p = \frac{s_g - s_l}{h_g - h_l} \tag{D.22}$$

$$\left(\frac{\partial s}{\partial p}\right)_h = \frac{ds_l}{dp} + \frac{dx}{dp}(s_g - s_l) + x\left(\frac{ds_g}{dp} - \frac{ds_l}{dp}\right) \tag{D.23}$$

where

$$\frac{ds_l}{dp} = \frac{cp_l}{T}\frac{dT}{dp} - \beta_l v_l \tag{D.24}$$

$$\frac{ds_g}{dp} = \frac{cp_g}{T}\frac{dT}{dp} - \beta_g v_g \tag{D.25}$$

Appendix E

Additional Component and System Models

The most important component models implemented in the new component model library are presented in chapter 4. Additional component models are introduced in chapters 6 and 7 when describing two demonstrating example applications. This chapter presents some additional component models.

E.1 The `StateViewerInterface`

In order to allow for the automated numbering of medium objects in simulations of closed cycles, a special component called `StateViewerInterface` is used that can be seen in figure 4.1 in front of the compressor. This component breaks the algebraic loop that would otherwise exist for the `index` that is defined in the fluid ports of the new component model library (see code listing 4.1).

```
model StateViewerInterface "Component to define starting point for index"
  FluidPort inlet "Inlet fluid port";
  FluidPort outlet "Outlet fluid port";
  parameter Integer startIndex = 0 "Start value for index";
equation
  inlet.m_flow + outlet.m_flow = 0 "Mass balance";
  inlet.H_flow + outlet.H_flow = 0 "Energy balance";
  inlet.p = outlet.p "Momentum balance";

  outlet.index = startIndex;

  ... // further code omitted
end BaseComponent;
```

Code Listing E.1: Code for `StateViewerInterface`.

E.2 Model for Prototype Peltier Heat Exchanger

A model for a prototype Peltier heat exchanger combining component models from the new component model library and from the Modelica Standard Library with additional new component models is presented in chapter 6. Figure E.1 shows the velocity vectors within a single channel as shown in figure 6.3 obtained from a CFD simulation carried out using FLUENT 6.0. For the CFD simulation, an inlet volume flow rate of $\dot{V} = 1\ \frac{l}{min}$ and a temperature $\vartheta = 20\ °\mathrm{C}$ were chosen.

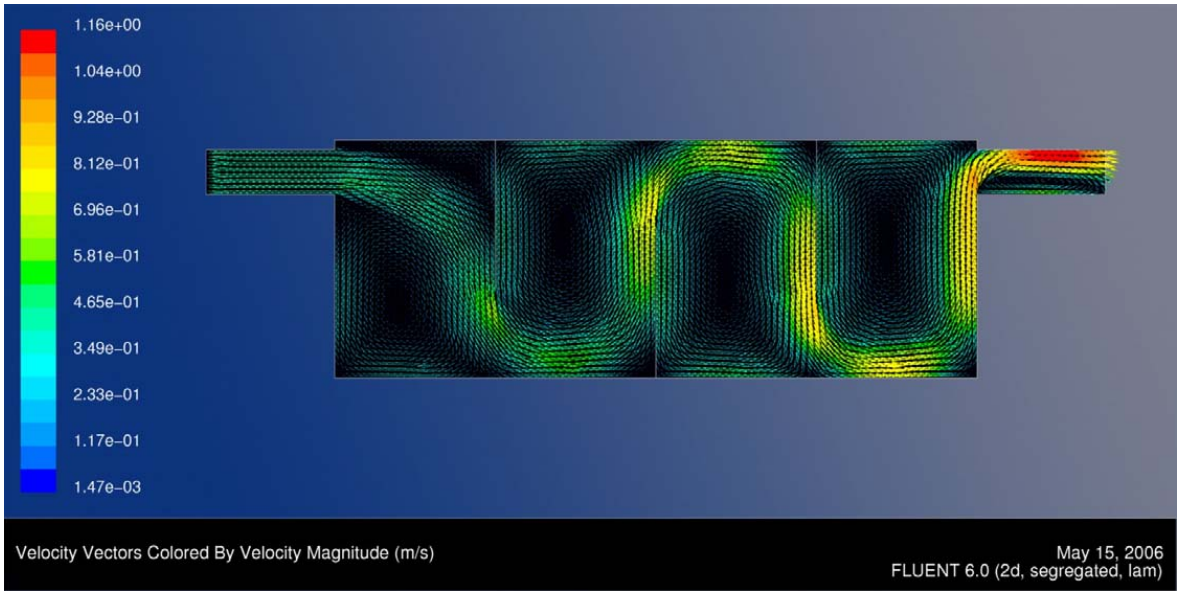


Figure E.1: Numerical results from CFD simulation of water flow through single aluminium channel.

For the computation of the coefficient of heat transfer α, an average velocity of $w_m = 0.7\ \frac{m}{s}$ is assumed according to the results shown in figure E.1. The fluid properties of water at $\vartheta = 20\ °\mathrm{C}$ are

$$\varrho(20°C) = 998\ \frac{kg}{m^3} \qquad \nu(20°C) = 1.003 \cdot 10^{-6}\ \frac{m}{s}$$
$$c_p(20°C) = 4184\ \frac{J}{kg \cdot K} \qquad \lambda(20°C) = 589.5 \cdot 10^{-3}\ \frac{W}{m \cdot K}$$

From these properties, the Reynolds and the Prandtl number can be computed to be

$$Re = \frac{w_m\ d_h}{\nu} = 48,853$$
$$Pr = \frac{\nu\ \varrho\ c_p}{\lambda} = 7.00$$

using a hydraulic diameter $d_h = 0.07\ m$. From this, the Nusselt number can be computed (see for example Köhler, 2007b)

$$Nu = f_3 \frac{\frac{\xi}{8}(Re - 1000)Pr}{1 + 12.7\sqrt{\frac{\xi}{8}}(Pr^{2/3} - 1)} \left\{1 + \left(\frac{d_h}{L}\right)^{\frac{2}{3}}\right\} = 481.99$$

where the length is $L \approx 200\ mm$, $f_3 = 1.0$, and

$$\xi = (1.82\ \log_{10}(Re) - 1.64)^{-2} = 0.021$$

The coefficient of heat transfer α can now be computed to be

$$\alpha = \frac{Nu\ \lambda}{d_h} = 4121\ \frac{W}{m^2 \cdot K}$$

Figure E.2 shows the complete system setup in Dymola to simulate the prototype Peltier water-water heat exchanger using components models from various libraries.

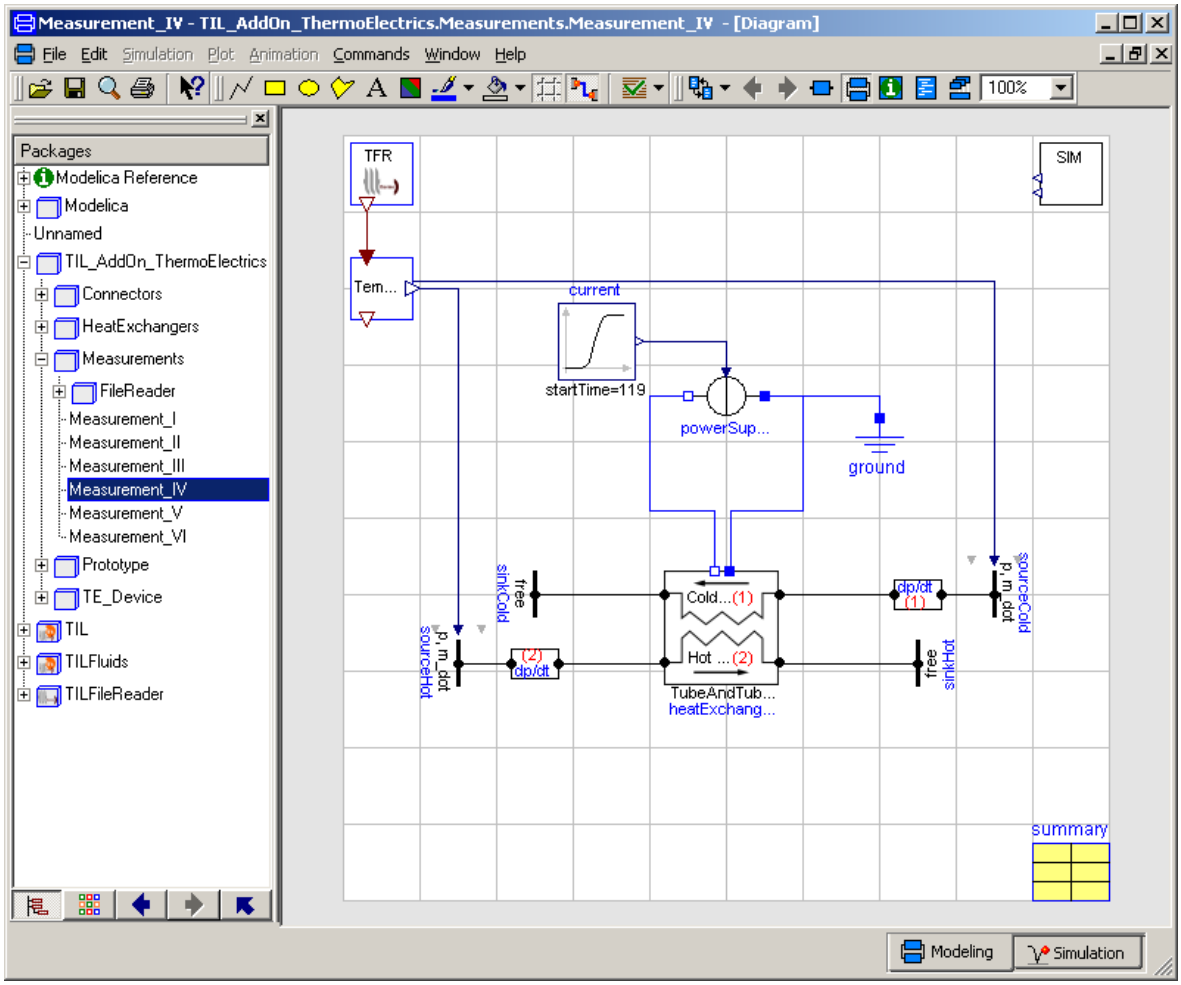


Figure E.2: Setup of simulation for tube-and-tube Peltier heat exchanger in Dymola.