H. Tummescheit, J. Eborn, F.J. Wagner:
Development of a Modelica Base Library for Modeling of Thermo-Hydraulic Systems.
Modelica Workshop 2000 Proceedings, pp. 41-51.

Paper presented at the Modelica Workshop 2000, Oct. 23.-24., 2000, Lund, Sweden.

All papers of this workshop can be downloaded from
http://www.Modelica.org/modelica2000/proceedings.html

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

Hubertus Tummescheit[†], *Jonas Eborn*[†] *and Falko Jens Wagner*[‡]

[†]Department of Automatic Control
Lund University, Sweden
{hubertus,jonas}@control.lth.se

[‡]Department of Energy Engineering
Technical University of Denmark
falko@et.dtu.dk

Abstract

This paper presents current results of an ongoing project to develop a Modelica base library for thermo-hydraulic systems.

There are many different aspects to the development of such a library, from the basic physics of fluids and heat to the structuring of model classes in the library and the actual implementation in the Modelica language. The structuring should define interfaces and partial classes that facilitate reuse to make the library general and easy to use. Different choices of media, use of different state variables as well as different levels of complexity in modeling is anticipated in the library structure.

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

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

1. Introduction

With the modeling language Modelica™, (3, 5), it is possible to create model libraries for different application areas. In the current Modelica base library distribution there are libraries for multi-body systems, electrical systems and block diagrams. To further expand the range of applications for Modelica a base library for modeling and simulation of thermo-hydraulic systems is also needed. A thermo-hydraulic base library should cover the basic physics of flows of fluids and heat. It also needs to cover models for properties of fluids like water and refrigerants. The library would then be useful in several application areas, e. g., power generation plants, district heating and refrigeration systems.

The general goal of the library is to provide a framework and basic building blocks for modeling thermo-hydraulic systems in Modelica. For obvious reasons it is impossible to provide components for every application, so one of the main goals is extensibility. For the same reason, much more emphasis will be put on the basic parts of the library, such as medium mod-

els and essential control volumes, than on an exhaustive application library. The focus of the library is on models of homogeneous one- and two-phase flows, non-homogeneous and multiphase flows are not taken into account yet. It is necessary to support bidirectional flow, because flow directions can change during simulation or are not known initially in networks.

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

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

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

Some of the ideas for the thermo-hydraulic base library have previously been presented in (2, 8, 9, 10) as well as object-oriented component based modeling has been presented in (11, 12).

2. Basic ideas

The basic design principles of the library are:

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

The first guideline puts a constraint on the discretization method used in the distributed parameter models: only the “staggered grid” or finite volume method (see Figure 1), where all fluxes are calculated on the border of a control volume and the intensive quantities are calculated in the center of a control volume, reduces to a useful model in the lumped parameter case. The finite volume method (6) is common for systems modeling with one-dimensional discretizations, but has the drawback that spatial derivatives are only first order accurate.

The ability to handle reversing flows requires extra information in the connectors between models. Transported properties, e.g., enthalpy and composition, would need to be included twice, upstream and downstream. This has instead been solved by including convective heat flow and component mass flows in the connectors. Thus the information needed for the balance equations (see Section 3.1) is contained in variables depending on the flow direction, i.e., mass flow and convective heat

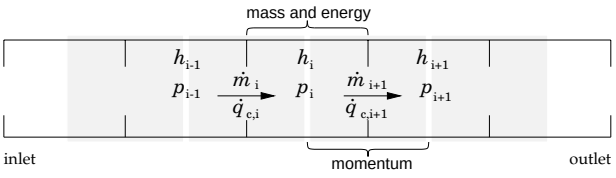

Fig. 1 Discretized Flow Grid

flow. In contrast the transported properties in the connector are always taken from the closest control volume.

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

$$\{p, h, \dot{m}, \dot{q}_c, \rho, T, s, \kappa\}$$

where $\dot{m}$ is mass flow, $\dot{q}_c$ is convective heat flow and κ is the ratio of specific heats.

3. Control volume equations

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

For a complete model description of a control volume, three parts are needed:

- Balance equations (mass, energy and momentum)
- Constitutive equations (pressure drop, heat flow)
- Medium property routines

3.1 *Balance equations*

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

$$\frac{d}{dt}\begin{pmatrix} M \\ U \end{pmatrix} = \begin{pmatrix} \sum_i^n \dot{m}_i \\ \sum_i^n \dot{q}_{conv,i} + \sum_j^l \dot{q}_{transfer,j} \end{pmatrix} \tag{3.1}$$

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

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

Static flow models are much used in system simulation where the thermal behavior is the main concern. The dynamic momentum balance is useful for pressure wave propagation studies in a system which is mainly modeled with distributed parameter models. Keep in mind that it is possible to add other types of flow models to the existing structure, e.g., a momentum balance for variable cross-sectional area along the flow channel.

The dynamic momentum balance is used to calculate the mass flow rate,

$$\Delta z \frac{d\dot{m}}{dt} = \frac{dI}{dt} = \dot{I}_1 - \dot{I}_2 + (p_1 - p_2)A - F_{wall}$$

where $I = \dot{m}\Delta z$ is the momentum and the frictional force, F_{wall}, is given by some constitutive equation.

This alternating structure of one control volume and one flow model, see Figure 2, guarantees that the simulation problem is well specified and that there are no unnecessary algebraic loops. The models can be used in either lumped, compound models or distributed, vectorized models, giving the user a possibility to change the complexity of the system model.

3.2 *Constitutive equations*

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

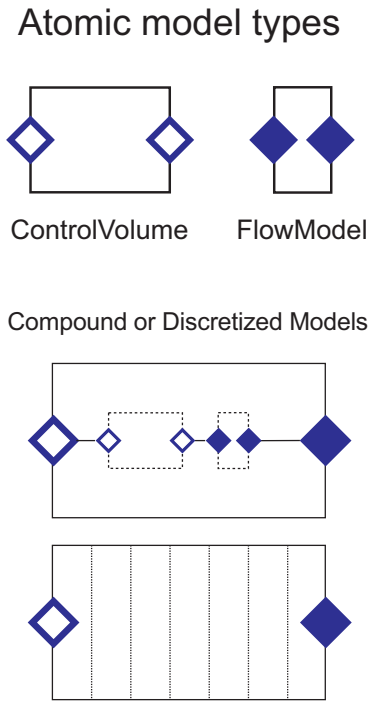

Fig. 2 Atomic, Compound and Discretized Models

These constitutive equations should be replaceable, in order to have a general model for a component that can be used in different situations by exchanging the model for the characteristics, see Section 4.1 for an example.

3.3 *Medium property routines*

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

- Accuracy
- Speed
- Robustness

In some areas there exist recommended formulations (IAPWS/IF97 for water (13)) or defacto standards (NIST-REFPROP routines for refrigerants, (4)) that have to be taken into account. External function call interfaces in Modelica make it possible to use these standards directly. Available routines and most medium property models in the literature (see, e. g., (7)) are designed with stationary calculations in mind, therefore they have to be extended to include some needed extra derivatives for dynamic calculations.

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

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

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

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

Summarizing this means that the medium properties that are provided with this library:

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

3.4 *State variable transformations*

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

The balance equations for mass and internal energy (3.1) can be rewritten into differential equations for ρ and u. A differentiation of $M = \rho V$ and $U = uM$ for a constant volume yields:

$$\begin{cases} V\dfrac{d\rho}{dt} &= \dfrac{dM}{dt} \\ M\dfrac{du}{dt} &= \dfrac{dU}{dt} - u\dfrac{dM}{dt} \end{cases} \tag{3.2}$$

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

$$\frac{d}{dt}\begin{pmatrix} \rho \\ u \end{pmatrix} = \underbrace{\begin{pmatrix} \left.\dfrac{\partial\rho}{\partial p}\right|_h & \left.\dfrac{\partial\rho}{\partial h}\right|_p \\ \left.\dfrac{\partial u}{\partial p}\right|_h & \left.\dfrac{\partial u}{\partial h}\right|_p \end{pmatrix}}_{\text{Jacobian, J}} \frac{d}{dt}\begin{pmatrix} p \\ h \end{pmatrix} \tag{3.3}$$

To obtain differential equations for pressure and enthalpy eq. (3.3) must be solved for the derivative of (p, h)

$$\frac{d}{dt}\begin{pmatrix} p \\ h \end{pmatrix} = \mathsf{J}^{-1}\frac{d}{dt}\begin{pmatrix} \rho \\ u \end{pmatrix} \tag{3.4}$$

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

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

This gives the inverse Jacobian as

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

where a is the velocity of sound. By combining (3.4) and (3.2), multiplying with $M = \rho V$ and

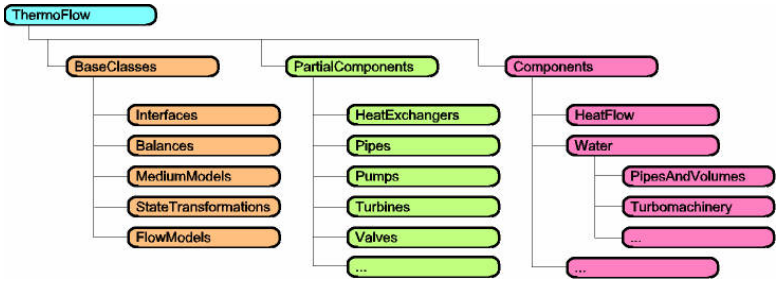

Fig. 3 Basic Package Structure of the ThermoFlow Library

noting that $h = u + p/\rho$ we obtain

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

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

3.5 *Library structure*

The main idea of the ThermoFlow library is to provide an extensible basis for a robust thermo-hydraulic component library. The structure of the library is divided into three parts, see Figure 3.

Base classes are the central part of models, the basic physical equations for a control volume and the connector types for flowing media; either single or multi-component, with a static or dynamic flow description.

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

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

4. Object-oriented modeling

Modelica is an object-oriented modeling language, designed for modeling physical systems. Many of the object-oriented features defined by (1) are found in the Modelica language:

- (Multiple) Inheritance
- Class parameterization
- Generalization

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

Class parameterization gives the possibility to implement generic classes that can be used for specialization later. With this, a parameter can be passed to a class during instantiation, giving the class the desired behavior, see Section 4.1.

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

4.1 *Object-oriented constructs in Modelica*

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

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

- Balance equations
- Flow model
- An empty shell with connectors

These subproblems are modeled individually in the following classes:

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

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

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

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

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

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

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

The GenericPressureLossModel does not have to contain anything, but for practical reasons (and as a base class for inheritance in specialized pressure loss models) it contains the most necessary variables.

During specialization in later classes, this generic pressure loss model is then replaced by a more meaningful model, containing not only the variables, but also some equations for calculating the actual pressure loss.

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

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

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

```
partial model FlowModelBase
  extends FlowVariables;
  extends TwoPort;
equation
```

```
  a.q_conv = if a_upstr
    then mdot*a.h
    else mdot*b.h;
end FlowModelBase;
```

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

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

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

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

4.2 *Summary*

Some examples have shown how important object-oriented constructs are implemented in the Modelica language. These constructs are used throughout the library structure (see Section 3.5) to facilitate wide spread use of generalization and code sharing and make the library more flexible.

5. Component models

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

5.1 *Pumps*

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

$$\Delta p_n = R_1 n_n + 2R_2 n_n V_n - R_3 |V_n| V_n \quad (5.1)$$

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

In terms of the ThermoFlow library structure, the pump can be modeled by using a lumped control volume and a lumped flow model according to Figure 2. The lumped flow model then represents the pump characteristic (5.1), whereas the lumped control volume in front of the pump is used according to the library structure. This control volume represents the volume of the pump, which should not be neglected.

5.2 *Heat exchangers*

A heat exchanger is modeled using base components from the library. Basically, it consists of two pipes connected by a heat conducting wall. Figure 4 shows the principle of modeling a heat exchanger and a sample system.

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

5.3 *Turbines*

Analogous to pumps, a turbine is modeled as a lumped control volume with a following lumped flow model. The flow model introduces the turbine characteristic or turbine profile.

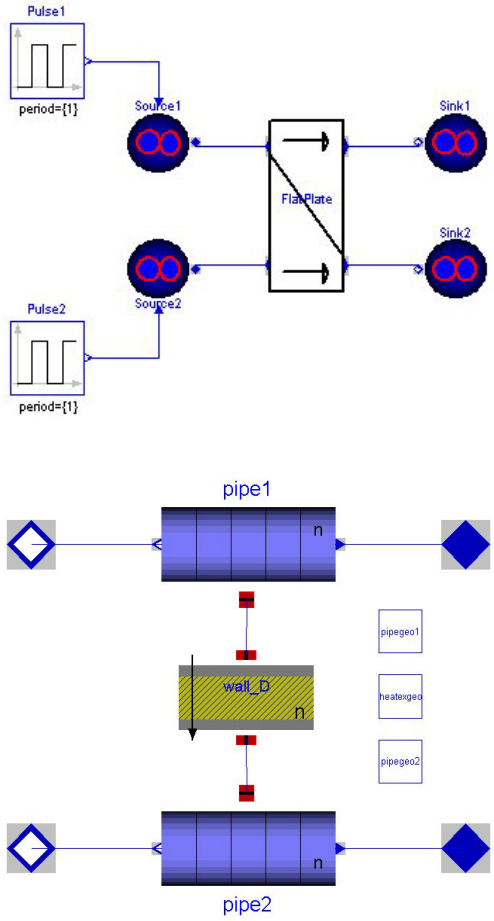

Fig. 4 Example system using a heat exchanger consisting of two pipes, a wall and 4 connectors

Currently two models are implemented in the base library. One model is according to Stodola, the other after Linnecken. The Stodola model is an idealized turbine with an infinite number of stages. The Linnecken model considers the maximum mass flow rate through the turbine stage and can be parameterized according to the type of the turbine and the number of stages.

5.4 *Reservoirs*

Some thermo-hydraulic systems are closed systems, e. g., power plants or refrigeration systems. But for general modeling purposes, sources and sinks are required. These are used to add "boundary" conditions to other components or systems of components. Typical sources are temperature, pressure or heat sources. They can either be fixed or depend on some input signal. The sample system in Figure 4 contains 2 controlled sources and 2 sinks.

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

Heat sources can either be fixed in temperature or fixed in heat flux, i. e., they can also be controlled by an external signal. A sink is usually just a fixed pressure control volume, and the assumption about the "infinite reservoir" holds as well.

6. Examples

A system with a parallel flow plate heat exchanger (see Figure 4) is simulated with a temperature increase on the hot side, followed by a pressure increase on the cold side with according mass flow rate increase. The result is shown in Figure 5.

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

Using the components implemented in the library, it is possible to build also more complex systems, e. g., power plants, see Figure 6.

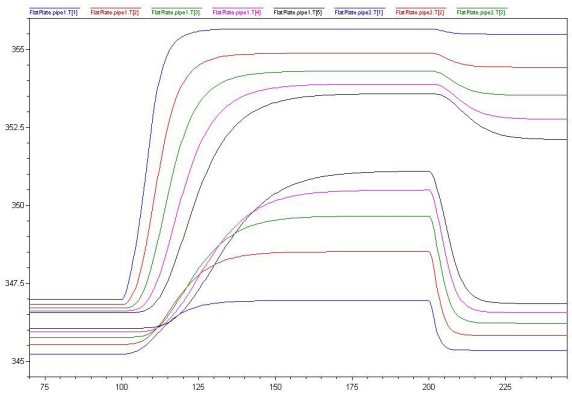

Fig. 5 Resulting temperatures from simulating the system in Figure 4

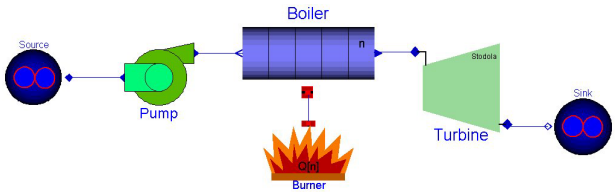

Fig. 6 Example system

7. Conclusion

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

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

Some further conclusions:

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

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

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

7.1 *Further information*

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

8. References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, New York, Berlin, 1996.

[2] J. Eborn, H. Tummescheit, and K. J. Åström. “Physical system modeling with

Modelica." In *14th World Congress of IFAC*, vol. N. IFAC, July 1999.

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

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

[5] Modelica Design Group. "The Modelica Language Specification." Version 1.3, `http://www.modelica.org/`, 1999.

[6] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation, 1980.

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

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

[9] H. Tummescheit. "Object-oriented modeling of physical systems, part 12." *Automatisierungstechnik*, **48:4**, 2000. In german.

[10] H. Tummescheit and J. Eborn. "Design of a thermo-hydraulic model library in Modelica." In Zobel and Moeller, Eds., *Proc. of the 12th European Simulation Multiconference, ESM'98*, pp. 132–136, Manchester, UK, June 1998. SCS.

[11] F. J. Wagner and M. Z. Poulsen. "C++ toolbox for object oriented modeling and dynamic simulation of physical systems." *SIMS Conference, Linkoeping*, 1999.

[12] F. J. Wagner, M. Z. Poulsen, P. G. Thomsen, and N. Houbaok. "Object oriented toolbox for modeling and simulation of dynamical systems." *SIAM Workshop*, 1998.

[13] W. Wagner and A. Kruse. *Properties of water and steam*. Springer, Berlin, 1998.