Proceedings
of the 3rd International Modelica Conference,
Linköping, November 3-4, 2003,
Peter Fritzson (editor)

Hilding Elmqvist, Hubertus Tummescheit and Martin Otter
Dynasim, Sweden; UTRC, USA; DLR, Germany:

Object-Oriented Modeling of Thermo-Fluid Systems

pp. 269-286

Paper presented at the 3rd International Modelica Conference, November 3-4, 2003, Linköpings Universitet, Linköping, Sweden, organized by The Modelica Association and Institutionen för datavetenskap, Linköpings universitet

All papers of this conference can be downloaded from
http://www.Modelica.org/Conference2003/papers.shtml

Object-Oriented Modeling of Thermo-Fluid Systems

Hilding Elmqvist[1], Hubertus Tummescheit[2], and Martin Otter[3]
[1]Dynasim AB, Lund, Sweden, www.dynasim.se, Elmqvist@dynasim.se
[2]UTRC, Hartford, U.S.A., Hubertus@control.lth.se
[3]DLR, Germany, www.robotic.dlr.de/Martin.Otter, Martin.Otter@dlr.de

Abstract

Modelica is used since 1998 to model thermo-fluid systems. At least eight different libraries in this field have been developed and are utilized in applications. In the last year the Modelica Association has made an attempt to standardize the most important interfaces, provide good solutions for the basic problems every library in this field have and supply sophisticated base elements, especially media descriptions. This paper summarizes the design, new Modelica language elements, new symbolic transformation algorithms and describes two new libraries – for media description and for fluid base components – that will be included in the Modelica standard library.

1 Introduction

Careful decomposition of a thermodynamic system is essential to achieve reusable components. This paper discusses appropriate Modelica interfaces to handle thermodynamic properties, empirical closure relations like pressure drop correlations, mass balances and energy balances. Special attention has been placed on allowing flows with changing directions and allowing ideal splitting and merging of flows by connecting several components at one junction as well as parallel flow paths having zero (neglected) volume. A purely declarative approach solves the problem of splitting and merging flows in a physically based way. For mixing, the resulting specific enthalpy or temperature is *implicitly* defined and is obtained by solving a system of equations.

All balance equations are provided in their natural form. Necessary differentiations are carried out by a tool through index reduction. Due to newly developed symbolic transformation algorithms, the described approach leads to the same simulation efficiency as previously developed thermo-fluid libraries, but without having their restrictions.

The discussed method is implemented in two new Modelica libraries, “Modelica_Fluid” and “Modelica_Media” that will become part of the free Modelica standard library as Modelica.Fluid and Modelica.Media. “Media” contains a generic interface to media property calculations with required and optional media variables. A large amount of pre-defined media models are provided based on media models of the ThermoFluid library Tummescheit and Eborn (2001). Especially, about 1200 gases and mixtures of these gases, as well as a high precision water model based on the IF97 standard are included. The “Fluid” library provides the generic fluid connectors and the most important basic devices, such as sources, sensors, and pipes for quasi 1-dimensional flow of media with single or multiple phases and single or multiple substances. The same device model is used for incompressible and compressible flow. A tool will perform the necessary equation changes by index reduction when, e.g., an incompressible medium model is replaced by a compressible one in a device model.

The “Fluid” and “Media” libraries are a good starting point for application specific libraries, such as for steam power plants, refrigeration systems, machine cooling, or thermo-hydraulic systems.

2 Devices, medium models, balance volumes and ports

We will consider thermodynamic properties of fluids in coupled devices, such as tanks, reactors, valves as well as pipes, Figure 1. Control volumes (or balance volumes) will be considered for all devices.

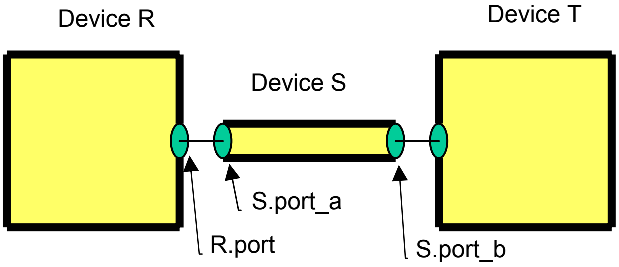


Figure 1. Connected devices

2.1 Medium models

The thermodynamic state of the fluid at any point is represented by two variables, e.g., pressure p and specific enthalpy h. Other thermodynamic quantities may be calculated from the chosen thermodynamic state variables. It is important that a model for a device can be written in such a way that it can be used in combination with different media models. This property is achieved by representing the media as a replaceable package. The details are given in Section 5. Such a media package contains, among other definitions, a model with three equations as shown in the following partial example for a simple model of air based on the ideal gas law:

```
package SimpleAir
  ...
  constant Integer nX = 0;
  model BaseProperties
    AbsolutePressure       p;
    Temperature            T;
    Density                d;
    SpecificInternalEnergy u;
    SpecificEnthalpy       h;
    MassFraction           X[nX];
    constant Real R_air = 287.0506;
    constant Real h0    = 274648.7;
  equation
    p = d*R_air*T;
    h = 1005.45*T + h0;
    u = h - p/d;
  end BaseProperties;
  ...
  end SimpleAir;
```

How such a media package can be utilized in a model is shown in the following heated device model without incoming or leaving mass flows.

```
model ClosedDevice
  import M = Modelica.Media;
  replaceable package Medium=
        M.Interfaces.PartialMedium;
  Medium.BaseProperties medium
  parameter …
equation
  // Mass balance
  der(m)  = 0;
  m = V*medium.d;

  // Energy balance
  der(U)  = Q;
  U = m*medium.u;
end ClosedDevice;
```

When using this device model, a specific medium has to be defined:

```
ClosedDevice device(redeclare
        package Medium = SimpleAir);
```

The device model is not influenced by the fact that the medium model is compressible or incompressible.

2.2 Ports

Figure 2 shows a detailed view of a connection between two devices. An important design decision is the selection of the Modelica connector that describes a device port. For the Modelica_Fluid library the connector is defined for quasi one-dimensional fluid flow in a piping network, with incompressible or compressible media models, one or more phases, and one or more substances. The connector variables are selected such that the equations of the connect(...) statements of connected components fulfill the following balance equations:

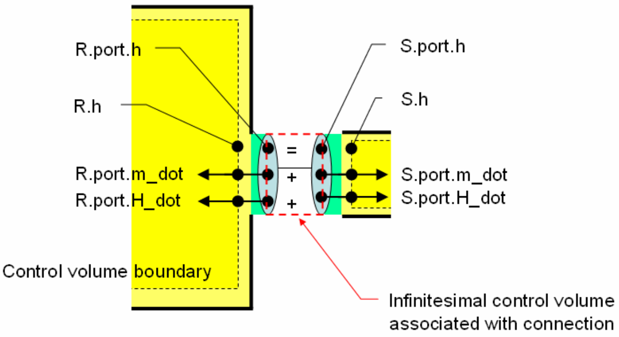


Figure 2. Details of device connection

- mass balance
- substance mass balance (of a medium with several substances).
- energy balance in the form of the "internal energy balance" (see Section 3).

Additionally, a non-redundant set of variables is used in the connector in order to not have any restrictions how components can be connected together (restrictions would be present, if an overdetermined set of describing variables would be used in the connector). These design requirements lead to a unique selection of variables in the connector:

Pressure p, specific (mixing) enthalpy h, independent (mixing) mass fractions X, mass flow rate m_dot, enthalpy flow rate H_dot, and the independent substance mass flow rates mX_dot

```
connector FluidPort
 replaceable package Medium =
 Modelica_Media.Interfaces.PartialMedium;

 Medium.AbsolutePressure  p;
 flow Medium.MassFlowRate m_dot;

 Medium.SpecificEnthalpy     h;
 flow Medium.EnthalpyFlowRate H_dot;

 Medium.MassFraction      X     [Medium.nX]
 flow Medium.MassFlowRate mX_dot[Medium.nX]
end FluidPort;
```

Due to the design of the connectors, the mass and energy balance is fulfilled in connection points (see also discussion of perfect mixing in the next Section). Since the momentum balance is not taken into account, device couplings with a considerable amount of losses (e.g., if pipes with different diameters are connected) have to be modeled with a dedicated loss model.

2.3 Splitting, Joining and Reverse Flow

Figure 2 also shows the control volumes associated with the devices and the boundary conditions. The flow through the port of a device is equal to the flow through the corresponding boundary of the control volume. Note that the specific enthalpy might have a discontinuity.

The connector variable FluidPort.h represents the specific enthalpy outside the control volume of the device. In fact, for two connected devices R and S, with FluidPort instances named “port”, R.port.h = S.port.h represent the specific enthalpy of an infinitesimally small control volume associated with the connection. The relation between the boundary and the port specific enthalpy depends on the flow direction. It is established indirectly by considering the enthalpy flow. We will introduce the notation h_{port} = R.port.h = S.port.h and will for simplicity of notation neglect spatial variation of the specific enthalpy, h_R and h_S, within each control volume. The enthalpy flow rate into device R, $\dot{H}_R$ is then dependent on the mass flow rate, $\dot{m}_R$ as follows.

$$\dot{H}_R = \begin{cases} \dot{m}_R h_{port} & \dot{m}_R > 0 \\ \dot{m}_R h_R & \text{otherwise} \end{cases}$$

This equation has to be present within the model of device R. Such conditional expressions could be written as if-then-else expressions, but to facilitate a recently identified set of powerful symbolic simplifications a new function, semiLinear(...), has been proposed for inclusion in the Modelica language (see also Figure 3), that can be used as follows in model R:

```
port.H_dot =
  semiLinear(port.m_dot, port.h, h);
```

The corresponding equation for a device S is

$$\dot{H}_S = \begin{cases} \dot{m}_S h_{port} & \dot{m}_S > 0 \\ \dot{m}_S h_S & \text{otherwise} \end{cases}$$

Devices R and S, see Figure 2, are connected together with a connect(...) statement of the form:

```
connect(R.port, S.port);
```

leading to the following *zero sum equations* that are equivalent to the mass and energy balance of the infinitesimal small control volume at the connection point:

$$0 = \dot{m}_R + \dot{m}_S$$
$$0 = \dot{H}_R + \dot{H}_S$$

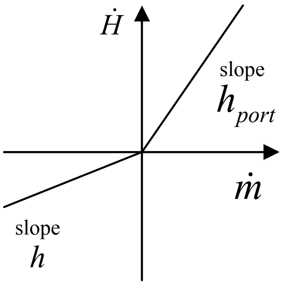


Figure 3. The semiLinear(...) function

From these four equations, h_{port} can be solved

$$h_{port} = \begin{cases} h_S & \dot{m}_R > 0 \\ h_R & \dot{m}_R < 0 \\ \text{undefined} & \dot{m}_R = 0 \end{cases}$$

According to Modelica flow semantics, $\dot{m}_R > 0$ corresponds to flow *into* component R and therefore the specific enthalpy flowing across the boundary is h_S at the device boundary, h_{port}. It should be noted that although h_{port} is undefined for zero mass flow rate, $\dot{H}_R$ and $\dot{H}_S$ are well-defined as zero, i.e., the dynamics of the system are independent of what value is chosen for h_{port}.

We will now consider the connection of three ports R.port, S.port and T.port. A symbolic solution of the common specific enthalpy,

h = R.port.h = S.port.h = T.port.h

is given by

```
h = -(
  (if R.port.m_dot > 0 then 0 else
    R.port.m_dot*R.h)+
  (if S.port.m_dot > 0 then 0 else
    S.port.m_dot*S.h)+
  (if T.port.m_dot > 0 then 0 else
    T.port.m_dot*T.h) )
  / (
  (if R.port.m_dot > 0 then
    R.port.m_dot else 0)+
  (if S.port.m_dot > 0 then
    S.port.m_dot else 0)+
  (if T.port.m_dot > 0 then
    T.port.m_dot else 0) )
```

For a *splitting* flow, for example from R to S and T, i.e., `R.port.m_dot < 0`, `S.port.m_dot > 0` and `T.port.m_dot > 0`, we get

```
h = -R.port.m_dot*R.h /
  (S.port.m_dot + T.port.m_dot )
```

Since

```
0 = R.port.m_dot + S.port.m_dot +
     T.port.m_dot
```

the specific enthalpy h in the port is computed as

```
h=-R.port.m_dot*R.h/(-R.port.m_dot )
```

or

```
h = R.h
```

For a *merging* flow, for example, from R and T into S (i.e., R.port.m_dot < 0, S.port.m_dot < 0 and T.port.m_dot > 0) we get

```
h = -(R.port.m_dot*R.h +
    S.port.m_dot*S.h) / T.port.m_dot
```

or

```
h=(R.port.m_dot*R.h+S.port.m_dot*S.h)
  /(R.port.m_dot + S.port.m_dot)
```

i.e., the perfect mixing condition.

The degenerate case that all mass flows are zero can be handled symbolically by the tool, as it does not influence the dynamics: For two connected devices R and S, the division with R.port.m_dot can be performed symbolically leading to

```
h = if R.port.m_dot > 0 then R.h
                         else S.h
```

As a result, for zero mass flow rate h = S.h. For three and more connected devices, the equation system is underdetermined. From the infinitely many solutions the one can be picked that is closest to the solution in the previous integrator step.

It should be noted that a similar approach could be used to handle flow composition for flows with several substances.

Earlier attempts tried to solve a restricted problem of changing flow direction in a programming style, i.e., by *explicitly* defining the temperature depending on the flow direction. Such a method cannot be generalized to mixing flows, because the temperature is not given by equations in just one volume. The presented solution for splitting and joining flows is derived by considering the equations of a small connection volume. By setting it's mass to zero, the usual sum-to-zero equations for mass flow rate and energy flow rate are obtained. This means that the usual flow semantics is appropriate for modeling of splitting and merging flows.

3 Mass-, momentum- and energy-balances

We will show a general implementation of the governing equations, which might serve as a template for specialized models. Consider the equations (mass, momentum and energy balances) for quasi-one-dimensional flow in a device with flow ports in the ends such as a pipe, Thomas (1999) [16], Anderson (1995) [1].

$$\frac{\partial(\rho A)}{\partial t}+\frac{\partial(\rho A v)}{\partial x}=0$$

$$\frac{\partial(\rho v A)}{\partial t}+\frac{\partial(\rho v^2 A)}{\partial x}=-A\frac{\partial p}{\partial x}-F_F-A\rho g\frac{\partial z}{\partial x}$$

$$\frac{\partial(\rho(u+\frac{v^2}{2})A)}{\partial t}+\frac{\partial(\rho v(u+\frac{p}{\rho}+\frac{v^2}{2})A)}{\partial x}=$$

$$-F_F v-A\rho v g\frac{\partial z}{\partial x}+\frac{\partial}{\partial x}(kA\frac{\partial T}{\partial x})$$

$$F_F=\frac{1}{2}\rho v|v|fS$$

where t represents time, x is the spatial coordinate along device, ρ is the density, v is the velocity, A is the area, p is the pressure, F_F represents the friction force per length, f is the Fanning friction factor, S is the circumference, g is the gravity constant, z is the vertical displacement, k is the thermal conductivity and medium properties:

$$p=p(\rho,T)$$

$$u=u(\rho,T)$$

$$h=u+p/\rho$$

where h is the specific enthalpy and u is the specific internal energy.

The energy equation can be considerably simplified by subtracting the momentum balance multiplied by v. Simplifications that are shown in the appendix, give the result.

$$\frac{\partial(\rho u A)}{\partial t}+\frac{\partial(\rho v(u+\frac{p}{\rho})A)}{\partial x}=vA\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}(kA\frac{\partial T}{\partial x})$$

Finite volume method

Such partial differential equations can be solved by various methods like finite difference, finite element or finite volume methods. The finite volume method is chosen because it has good properties with regards to maintaining the conserved quantities. The device is split into segments, for which the PDEs are integrated and approximated by ODEs. Let x=a and x=b be the coordinates for the ends of any such segment. Integrating the mass balance equation over the spatial coordinate, x, gives

$$\int_a^b \frac{\partial(\rho A)}{\partial t}dx + \rho A v\big|_{x=b} - \rho A v\big|_{x=a} = 0$$

Assuming the segment boundaries (a, b) to be constant, we can interchange the integral and derivative:

$$\frac{d(\int_a^b \rho A dx)}{dt} + \rho A v\big|_{x=b} - \rho A v\big|_{x=a} = 0$$

In order to handle the general case of changing volumes for, e.g., displacement pumps, tanks, or moving boundary models of two phase flows, this formula needs to be extended by use of the Leibnitz formula.

Introducing appropriate mean values for density and area and introducing incoming mass flow rates $\dot{m}$, i.e. $\dot{m}_b = -\rho A v\big|_{x=b}$ and $\dot{m}_a = \rho A v\big|_{x=a}$, we can rewrite the mass balance as:

$$\frac{d(\rho_m A_m (b-a))}{dt} = \dot{m}_a + \dot{m}_b$$

Introducing $m = \rho_m A_m L$ and $L = b - a$ gives the desired form of the mass balance

$$\frac{dm}{dt} = \dot{m}_a + \dot{m}_b$$

We proceed in a similar way with the momentum balance:

$$\int_a^b \frac{\partial(\rho v A)}{\partial t}dx + \rho v^2 A\big|_{x=b} - \rho v^2 A\big|_{x=a}$$

$$= -Ap\big|_{x=b} + Ap\big|_{x=a} + \int_a^b \frac{\partial A}{\partial x} p dx -$$

$$\int_a^b \frac{1}{2}\rho v|v| fS dx - \int_a^b A\rho g \frac{\partial z}{\partial x}dx$$

and introducing appropriate mean values gives:

$$\frac{d(\rho_m v_m A_m)}{dt}L + \rho v^2 A\big|_{x=b} - \rho v^2 A\big|_{x=a}$$

$$= A_m(p_a - p_b)$$

$$-\frac{1}{2}\rho_m v_m |v_m| f_m S_m L - A_m \rho_m g \Delta z$$

with $A_m = (A_a + A_b)/2$. Substitution by $\dot{m}$ and the values at the respective boundaries and introducing the approximation $\dot{m}_m = \frac{\dot{m}_a + \dot{m}_b}{2}$ gives

$$\frac{d\dot{m}_m}{dt}L = \frac{\dot{m}_a^2}{A_a p_a} - \frac{\dot{m}_b^2}{A_b p_b} + A_m(p_a - p_b)$$

$$-\frac{1}{2}\frac{1}{\rho_m A_m^2}\dot{m}_m|\dot{m}_m| f_m S_m L - A_m \rho_m g \Delta z$$

We will make the approximation that $\rho_a = \rho_b = \rho_m$ evaluated at mean pressure $p_m = \frac{p_a + p_b}{2}$.

Integrating the energy balance for internal energy gives:

$$\int_a^b \frac{\partial(\rho u A)}{\partial t}dx + \rho h v A\big|_{x=b} - \rho h v A\big|_{x=a} =$$

$$\int_a^b vA\frac{\partial p}{\partial x}dx + kA\frac{\partial T}{\partial x}\bigg|_{x=b} - kA\frac{\partial T}{\partial x}\bigg|_{x=a}$$

Substitution and approximation gives

$$\frac{d(\rho_m u_m A_m L)}{dt} - \dot{m}_b h_b - \dot{m}_a h_a =$$

$$v_m A_m(p_b - p_a) + k\frac{\partial T}{\partial x}\bigg|_{x=b} - k\frac{\partial T}{\partial x}\bigg|_{x=a}$$

Introducing $U = \rho_m u_m A_m L = m u_m$, the inner energy and $\dot{H} = \dot{m} \cdot h$, the enthalpy flow rate give

$$\frac{dU}{dt} = \dot{H}_a + \dot{H}_b + v_m A_m (p_b - p_a)$$

$$+ kA\frac{\partial T}{\partial x}\Big|_{x=b} - kA\frac{\partial T}{\partial x}\Big|_{x=a}$$

The diffusion term contains the temperature gradients at the segment boundaries. A first order approximation of the gradient is

$$\frac{\partial T}{\partial x}\bigg|_{x=a} = \frac{T(a+\frac{\Delta x}{2}) - T(a-\frac{\Delta x}{2})}{\Delta x}$$

It should be noticed that $T(a-\frac{\Delta x}{2})$ is a property of an adjacent segment, i.e. not directly accessible. However, such diffusion terms are already available in the model ThermalConductor of the Modelica.Thermal.HeatTransfer library. This means that we can introduce a heat flow port with T_m and $\dot{Q}$ and write the energy equation as

$$\frac{dU}{dt} = \dot{H}_a + \dot{H}_b + v_m A_m (p_b - p_a) + \dot{Q}$$

The flow variable $\dot{Q}$ will be the sum of the diffusion from neighboring segments at x=a and x=b and external heat transfer (for example in a heat exchanger).

Modelica model

The Modelica model equations corresponding to the mass- momentum and energy balances derived above are given below. In addition, a medium component is used for the mean quantities. The semiLinear function is used to handle the interfacing of the balance volume boundary quantities with the quantities of the device ports as discussed earlier.

```
model DeviceSegment
   replaceable package Medium =
   Modelica_Media.Interfaces.PartialMedium;
   FluidPort port_a (redeclare package
     Medium=Medium);
   FluidPort port_b (redeclare package
     Medium=Medium);
   Medium.BaseProperties medium;
   // Variable and parameter declarations
   equation
   // Mean values
   medium.p =(port_a.p + port_b.p)/2;
   m_dot_m = (port_a.m_dot-port_b.m_dot)/2;
   d_m = medium.d;

   // Mass balance
   der(m) = port_a.m_dot + port_b.m_dot;
   m = medium.d*A_m*L;

   // Substance balances
   port_a.mX_dot = semiLinear(port_a.m_dot,
     port_a.X, medium.X);
   port_b.mX_dot = semiLinear(port_b.m_dot,
     port_b.X, medium.X);
   der(mX) = port_a.mX_dot + port_b.mX_dot;
   mX = m*medium.X;

   // Momentum balance
   L*der(m_dot_m) =
     A_m*(port_a.p - port_b.p)
     + port_a.m_dot*port_a.m_dot/(A_a*d_m)
     - port_b.m_dot*port_b.m_dot/(A_b*d_m)
     - m_dot_m*abs(m_dot_m)/
      (2*d_m*A_m^2)*f*S*L
     - A_m*d_m*g*(Z_b - Z_a);

   // Energy balance
   port_a.H_dot = semiLinear(port_a.m_dot,
       port_a.h, medium.h);
   port_b.H_dot = semiLinear(port_b.m_dot,
       port_b.h, medium.h);
   der(U) = port_a.H_dot + port_b.H_dot +
     m_dot_m/d_m*(port_b.p - port_a.p) +
     heatPort.Q_dot;
   U = m*medium.u;
   heatPort.T = medium.T;
end DeviceSegment;
```

The model derivation given above is generic. It can be generalized and extended in many ways. For example, to allow changing volume of the segment, the integrations can be carried out with variable boundaries, using the Leibnitz rule. In the above derivations, simple definitions of the mean values were used. It is possible to get better accuracy, for example, by using an upwind scheme taking into account the flow direction when calculating the mean values.

A staggered grid is sometimes used for solving such PDEs. It is claimed to give better convergence properties in certain cases by a better approximation of the pressure gradient. It is possible to make such an implementation in Modelica. In fact, the ThermoFluid library uses the staggered grid approach. In this case, the equation for momentum is integrated over another interval $\left[a-\frac{L}{2}, a+\frac{L}{2}\right]$. This momentum can be included in a flow element model. The mass and energy balances are included in a finite volume model. There are special problems of communicating, for example, the momentum term $\rho v^2 A\Big|_{x=a+L/2} - \rho v^2 A\Big|_{x=a-L/2}$ since the flow element is assumed to have the same mass flow rate at both its connectors. Additional, non-physical, connectors or additional connector variables need to be introduced in order to communicate these variables to neighboring flow elements.

4 Pressure Loss due to Friction

The momentum balance contains a term for the friction force

$$F_{fric} = \frac{1}{2}\frac{1}{\rho_m A_m^2}\dot{m}_m\left|\dot{m}_m\right| f_m S_m L$$

Often, the pressure loss is used instead of the friction force ($p_{Loss} = F_{fric}/A_m$) and different equations are in use to compute the pressure loss from the mass flow rate. In the Modelica_Fluid library a component to model this pressure loss is available that provides two versions of a generic pressure loss equation:

```
if from_dp then
  $\dot{m}_m = f_1(p_{Loss},...)$
else
  $p_{Loss} = f_2(\dot{m}_m,...)$
end if
```

Using the parameter "from_dp" in the "Advanced"-menu, users can select whether the mass flow rate is computed from the pressure loss (this is the default) or whether the pressure loss is computed from the mass flow rate. Depending on how the device is connected in a network, there might be fewer non-linear equations if parameter "from_dp" is selected correspondingly. In a future version, this selection might be performed automatically by a tool.

The user can currently choose between three variants of the pressure loss model:

1. **Constant Laminar**: $p_{Loss} = k \cdot \dot{m}$

   It is assumed that the flow is only laminar. The constant k is defined by providing p_{Loss} and $\dot{m}$ for nominal flow conditions that, for example, are determined by measurements.
2. **Constant Turbulent**: $p_{Loss} = k \cdot \dot{m} \cdot |\dot{m}|$.

   It is assumed that the flow is only turbulent. Again, the constant k is defined by providing p_{Loss} and $\dot{m}$ for nominal flow conditions. For small mass flow rates, the quadratic, or in the inverse case the square root, characteristic is replaced by a cubic polynomial. This avoids the usual problems at small mass flow rates.
3. **Detailed Friction**: provides a detailed model of frictional losses for commercial pipes with non-uniform roughness (including the smooth pipe as a special case) according to.:

$$p_{Loss} = \lambda(\mathrm{Re},\Delta) \cdot \frac{L}{2D} \cdot \rho \cdot \mathrm{v} \cdot |\,\mathrm{v}\,|$$

$$= \lambda_2(\mathrm{Re},\Delta) \cdot \frac{L\eta^2}{2D^3\rho^3} = \lambda_2(\mathrm{Re},\Delta) \cdot k_2$$

$$\mathrm{Re} = \frac{\mathrm{v} \cdot \mathrm{D} \cdot \rho}{\eta} = \frac{D}{A \cdot \eta} \cdot \dot{m}$$

with

λ : friction coefficient (= 4·f_m)
λ_2 : used friction coefficient (= λ·Re·|Re|)
Re: Reynolds number.
L : length of pipe
A : cross-sectional area of pipe
D : hydraulic diameter of pipe
= 4*A/wetted perimeter
(circular cross Section: D = diameter)
δ : Absolute roughness of inner pipe wall
(= averaged height of asperities)
Δ : Relative roughness (=δ/D)
ρ : density
η : dynamic viscosity
v : Mean velocity
k_2 abbreviation for $L\eta^2/(2D^3\rho^3)$

Note that the Reynolds number might be negative if the velocity or the mass flow rate is negative. The "Detailed Friction" variant will be discussed in more detail, since several implementation choices are non-standard: The first equation above to compute the pressure loss as a function of the friction coefficient λ and the mean velocity v is usually used and presented in textbooks, see **Figure 4**. This form is **not** suited for a simulation program since λ = 64/|Re| if |Re| < 2000, i.e., a division by zero occurs for zero mass flow rate because Re = 0 in this case. More useful for a simulation model is the friction coefficient λ_2 = λ·Re·|Re| introduced for the pipe loss component, because λ_2 = 64·Re if Re < 2000 and therefore no problems for zero mass flow rate occur. The characteristic of λ_2 is shown in **Figure 5** and is implemented in the pipe loss model. The absolute roughness δ of the pipe is a parameter of this model.

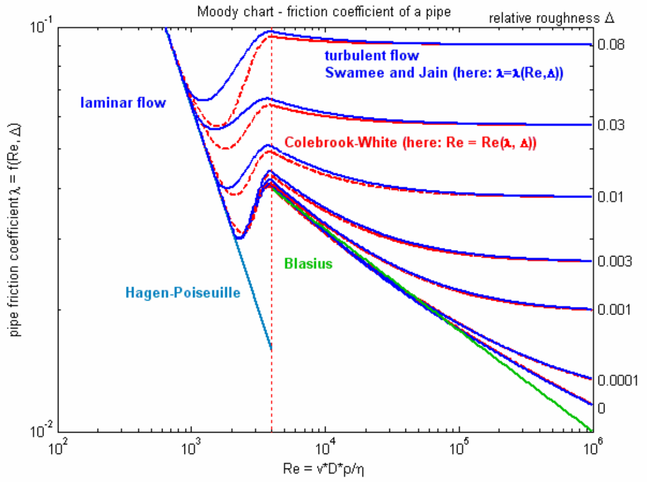


Figure 4. Moody Chart: lg(λ) = f (lg(Re), Δ)

The pressure loss characteristic is divided into three regions:

Region 1: For **Re ≤ 2000**, the flow is **laminar** and the exact solution of the 3-dim. Navier-Stokes equations (momentum and mass balance) is used under the assumptions of steady flow, constant pressure gradient and constant density and viscosity (= Hagen-Poiseuille flow):

$$\lambda_2 = 64 \cdot \mathrm{Re} \quad \text{or} \quad p_{Loss} = \frac{64 \cdot k_2 \cdot D}{A \cdot \eta} \cdot \dot{m}$$

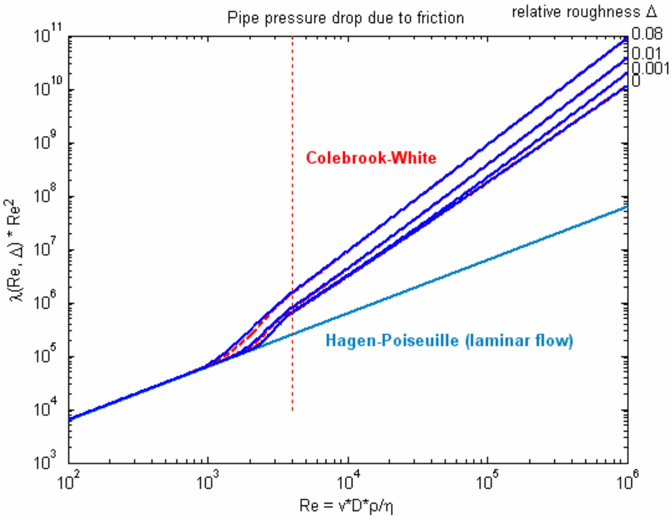


Figure 5. $\lambda_2 = \lambda_2(\mathrm{Re}, \Delta)$ = λ·Re·|Re|.
(x-axis: lg(Re), y-axis: lg(λ_2))

Region 3: For **Re ≥ 4000**, the flow is **turbulent**. Depending on parameter "`from_dp`" either of two explicit equations are used: If `from_dp = true` ($\dot{m} = f_1(p_{Loss})$), λ_2 is computed directly from p_{Loss} using $\lambda_2 = p_{Loss}/k_2$. The Colebrook-White equation (Colebrook (1939); Idelchik (1994) p. 83, eq. (2-9))

$$\frac{1}{\sqrt{\lambda}} = -2 \cdot \lg\left(\frac{2.51}{\text{Re}\sqrt{\lambda}} + 0.27 \cdot \Delta\right)$$

gives an implicit relationship between Re and λ. Inserting $\lambda_2 = \lambda \cdot \text{Re} \cdot |\text{Re}|$ allows to solve this equation analytically for Re:

$$\text{Re} = -2 \cdot \sqrt{|\lambda_2|} \lg\left(\frac{2.51}{\sqrt{|\lambda_2|}} + 0.27 \cdot \Delta\right) \cdot \text{sign}(\lambda_2)$$

These are the full-line curves in **Figure 4** and **Figure 5**. If `from_dp = false` ($p_{Loss} = f_2(\dot{m})$), λ_2 is computed by an approximation of the inverse of the Colebrook-White equation (Swamee and Jain (1976); Miller (1990) p. 191, eq. (8.4)) adapted to λ_2:

$$\lambda_2 = 0.25 \cdot \left[\text{Re}/\lg\left(\frac{\Delta}{3.7} + \frac{5.74}{|\text{Re}|^{0.9}}\right)\right]^2 \cdot \text{sign}(\text{Re})$$

These are the dotted-line curves in **Figure 4** and **Figure 5**.

Region 2: For **2000 ≤ Re ≤ 4000** there is a **transition** region between laminar and turbulent flow. The value of λ_2 depends on more factors than just the Reynolds number and the relative roughness, therefore only crude approximations are possible in this area. A laminar flow up to Re = 2000 is only reached for smooth pipes. The deviation Reynolds number Re1 at which the transition region starts is computed according to (Idelchik (1994), p. 81, sect. 2.1.21):

$$\text{Re1} = 745 \cdot e^{k3}$$
$$k3 = \textbf{if } \Delta \le 0.0065 \textbf{ then } 1 \textbf{ else } 0.0065/\Delta$$

Between Re1 = Re1(Δ) and Re2 = 4000, λ_2 is approximated by a cubic polynomial in the "lg(λ_2) = f(lg(Re))" chart (see **Figure 5**) such that the first derivative is continuous at these two points. In order to avoid the solution of non-linear equations, two different cubic polynomials are used for the direct and the inverse formulation (from_dp = **false**/**true**). This leads to some discrepancies in λ and λ_2 if Δ > 0.003 (= differences between the full and the dotted curves in the above Figures). This is acceptable, because the transition region is not precisely known since the actual friction coefficient depends on additional factors and since the operating points are usually not in this region.

The pressure loss equations above are valid for incompressible flow. According to (Idelchick (1994) p. 97, sect. 2.1.81) they can also be applied for compressible flow up to a Mach number of about Ma = 0.6 with a maximum error in λ of about 3 %. In a wide region the effect of gas compressibility can be taken into account by:

$$\lambda_{comp} = \lambda \cdot \left(1 + \frac{\kappa - 1}{2} \cdot \text{Ma}^2\right)^{-0.47}$$

where κ is the isentropic coefficient (for ideal gases, κ is the ratio of specific heat capacities c_p/c_v). This effect is not yet included in the pipe friction model. Another restriction is that the pressure loss model is valid only for steady state or slowly changing mass flow rate. For large fluid acceleration, the pressure drop depends additionally on the frequency of the changing mass flow rate.

To summarize, the pipe friction component provides a detailed pressure loss model in pipes in the form $\dot{m} = f_1(p_{Loss}, \Delta)$ or $p_{Loss} = f_2(\dot{m}, \Delta)$. These functions are continuous and differentiable, are provided in an explicit form without solving non-linear equations, and do behave well also at small mass flow rates. This pressure loss model can be used stand-alone in a static momentum balance and in a dynamic momentum balance as the friction pressure drop term. It is valid for incompressible and compressible flow up to a Mach number of 0.6.

5 Standard Medium Interface

The main properties of a single substance medium are described by 3 algebraic equations between the 5 thermodynamic variables pressure (p), temperature (T), density (d), specific internal energy (u) and specific enthalpy (h). In a medium model, three of these variables are given as function of the remaining two. For multiple substance media, additionally nX independent mass fractions X[nX] are present. For example, if p and T are selected as independent variables besides X, a medium model provides the algebraic equations

$$d = d(p, T, X)$$
$$u = u(p, T, X)$$
$$h = h(p, T, X)$$

The mass and energy balance equations in a device structurally have the following form for a single substance medium (see Section 3):

$$m = d \cdot V$$

$$U = m \cdot u$$

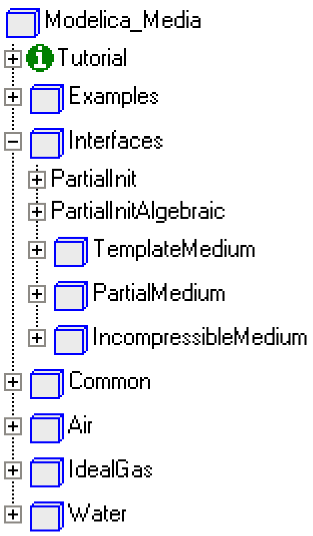
$$\frac{dm}{dt} = \sum_i \dot{m}_i \quad \text{// mass balance}$$

$$\frac{dU}{dt} = \sum_i \dot{m}_i h_i + \sum Q \quad \text{// energy balance}$$

where m is the mass and U is the internal energy in the control volume. Since the time derivatives of m and U appear, the derivatives of density d and internal energy u are implicitly needed which in turn means that the partial derivatives of d(p,T) and u(p,T) with respect to the independent variables p and T have to be calculated. As a result, the balance equations are reformulated in the variables p, T and this requires differentiation and formula manipulation.

Depending on the modeled device, additional fluid properties are needed, e.g., the dynamic viscosity if friction is modeled directly or the thermal conductivity for heat transfer coefficients or if diffusion is taken into account. Finally, a fluid may undergo phase changes and/or multiple substances may be involved.

Obtaining and computing the discussed fluid properties often takes the most effort in the modeling process. The availability of measurement data or correlations defines the level of accuracy that can be obtained with a thermo-fluid model. The needs of applications vary broadly from very simple properties with constant density and constant heat capacity to highly accurate non-linear models.



In order to ease fluid modeling with Modelica, a free Modelica library has been developed that provides (a) a standardized interface to media models and (b) a growing number of at once useable media models based on this interface, see Figure on the left. The temporary name of this library is “Modelica_Media”. It is planned to include this package in the Modelica standard library as Modelica.Media after an evaluation phase.

The Modelica_Media library is designed such that it can be used in different thermo-fluid libraries that may, e.g., have completely different connector definitions and design philosophies. In particular, the Modelica_Fluid library discussed in previous sections is based on this library, but it might also be useful for other thermo-fluid libraries. The Modelica_Media library has the following fundamental properties:

- Different independent medium variables may be used for media description, e.g., p,T or p,h or d,T or p,d.
- The definition of the medium is decoupled from the formulation of the balance equations in order that the balance equations can be formulated in their most natural form. There is enough information available for a tool to transform the medium equations into the form needed by the balance equations. This is achieved with the same efficiency as a usually used balance equation dedicated to a particular set of independent medium variables.
- Device models can be implemented independently of the choice of medium model. For example, exchanging an incompressible by a compressible medium model or a single by a multiple substance medium model is usually possible and has no major influence on the design of the device model.

5.1 Structure of Medium Interface

A medium model of Modelica_Media is essentially a **package** that contains the following definitions (the basic idea for this approach is from Newman et al (2002)):

- Definition of **constants**, such as the medium name or the number of substances.
- A **model** in the package that contains the 3 basic thermodynamic equations that relate the 5+nX primary medium variables.
- **Optional functions** to compute medium properties that are only needed in certain circumstances, such as dynamic viscosity. These optional functions need not be provided by every medium model.
- **Type** definitions, which are adapted to the particular medium. For example, a type “Temperature” is defined where the attributes “min” and “max” define the validity region of the medium. In a device model, it is advisable to use these type definitions, e.g., for parameters, in order that medium limits are checked as early as possible.

Note, although we use the term “medium model”, this is actually a Modelica “package” that contains all the constants and definitions required for a complete “medium model”. The basic interface to a medium is defined by Modelica_Media. Interfaces.PartialMedium that has the following structure:

```
partial package PartialMedium
  import SI = Modelica.SIunits;

  constant String  mediumName;
  constant String  substanceNames;
  constant Boolean incompressible;
  constant Boolean reducedX;
  constant Integer nX = size(
                   substanceNames,1);

  record BasePropertiesRecord
    AbsolutePressure       p;
    Temperature            T;
    Density                d;
    SpecificInternalEnergy u;
    SpecificEnthalpy       h;
    MassFraction           X[nX];
  end BasePropertiesRecord;

  replaceable model BaseProperties
    extends BasePropertiesRecord;
    // parameter declarations
  end BaseProperties;

  // optional medium properties
  replaceable partial function
                   dynamicViscosity
    input  BasePropertiesRecord
                            medium;
    output DynamicViscosity eta;
  end dynamicViscosity;
  // other optional functions

  // medium specific types
  type AbsolutePressure =
         SI.AbsolutePressure (
                min     = 0,
                max     = 1.e8,
                nominal = 1.e5,
                start   = 1.e5);
  type DynamicViscosity = ...;
  // other type definitions
end PartialMedium;
```

We will discuss all parts of this package in the following paragraphs. An actual medium model should extend from PartialMedium and has to provide implementations of the various parts.

The constants at the beginning of the package (with exception of nX) do not have a value yet (this is valid in Modelica), but a value has to be provided when extending from package PartialMedium. Once a value is given, it cannot be changed any more. The reason to use constants instead of parameters in the model BaseProperties is that some of these constants have to be used in connector definitions (such as the number of mass fractions nX). When defining the connector, only *constants* in packages can be accessed, but not *parameters* in a model, because a connector cannot contain an instance of BaseProperties.

The record BasePropertiesRecord contains the variables primarily used in balance equations. Three equations for these variables have to be provided by every medium in model BaseProperties. Optional medium properties are defined by functions, such as the function "dynamicViscosity" (see code Section above) to compute the dynamic viscosity. Model BaseProperties extends from the record and the optional functions have an instance of this record as an input argument. This construction simplifies the usage considerably as demonstrated in the following code fragment:

```
replaceable package
             Medium = PartialMedium;
Medium.BaseProperties   medium;
Medium.DynamicViscosity eta;
  ...
U  =m*medium.u; //Internal energy
eta=Medium.dynamicViscosity(medium);
```

"Medium" is the medium package that satisfies the requirements of a "PartialMedium" (when using the model above, a value for Medium has to be provided by a redeclaration). The "medium" component is an instance of the model "Medium.BaseProperties" and contains the core medium equations. Variables in this model can be accessed just by dot-notation, such as medium.u or medium.T. If an optional medium variable has to be computed, the corresponding function from the actual Medium package is called, such as "Medium.dynamicViscosity". The medium instance can be given as input argument to this function, because model Medium.BaseProperties is a subclass of BasePropertiesRecord – the argument required from the function.

If a medium model does not provide implementations of all optional functions and one of these functions is called in a model, an error occurs during translation since the not redeclared optional functions have the "partial" attribute. For example, if function dynamicViscosity is not provided in the medium model when it is used, only simple pressure drop loss models without a reference to the viscosity can be used and not the sophisticated ones.

At the bottom of the PartialMedium package type declarations are present that are used in all other parts of the PartialMedium package and that should be used in all models and connectors where a medium model is accessed. The reason is that minimum, maximum, nominal and sometimes also start values are defined and these values can be adapted to the particular medium at hand. For example, the nominal value of AbsolutePressure is $1.0e^5$ Pa. If a simple model of water steam is used that is only valid above 100 °C, then the minimum value in the Temperature type should be set to this value. The minimum and maximum values are also important for parameters in order to get an early message if data outside of the validity region is given. The “nominal” attribute is important as a scaling value if the variable is used as a state in a differential equation or as an iteration variable in a non-linear system of equations. The “start” attribute is useful to provide a meaningful default start or guess value if the variable is used, e.g., as iteration variable in a non-linear system of equations. Note, all these attributes can be set specifically for a medium in the following way:

```
 package MyMedium
   extends PartialMedium(
      ...
      Temperature(min=373);
   );
   ...
end MyMedium;
```

The type PartialMedium.MassFlowRate is defined as

```
type MassFlowRate = SI.MassFlowRate
   (quantity =
      "MassFlowRate." + mediumName);
```

Note that the constant “mediumName”, that has to be defined in every medium model, is used in the quantity attribute. For example, if mediumName = “SimpleLiquidWater”, then the quantity attribute has the value “MassFlowRate.SimpleLiquidWater”. This type should be used in a connector definition of a fluid library:

```
 connector FluidPort
   replaceable package Medium =
                         PartialMedium;
   flow Medium.MassFlowRate m_dot;
   ...
end FluidPort;
```

In the model where this connector is used, the actual Medium has to be defined. Connectors can only be connected together, if the corresponding attributes are either not defined or have identical values. Since mediumName is part of the quantity attribute of MassFlowRate, it is not possible to connect connectors with different media models together. In Dymola this is already checked when models are connected together in the diagram layer of the graphical user interface.

5.2 Defining Medium Models

The definition of a new medium model based on the PartialMedium interface is demonstrated using a simple model for air. First, the template package “Modelica_Media.Interfaces.TemplateMedium” should be copied and renamed. Afterwards, all parts of this template should be adjusted to the actual medium model. In particular:

```
 package SimpleAir
   extends Modelica_Media.Interfaces.
                        PartialMedium(
    mediumName = "SimpleAir";
    substanceNames  = fill("",0);
    incompressible  = false;
    reducedX        = true;
   );
   ...
end SimpleAir;
```

The new medium package is extended from PartialMedium and all constants that do not have a value in PartialMedium are defined now. If the medium consists of only one substance, set the dimension of the substanceNames vector to zero with the **fill**(..) operator. If the medium defines the density to be a constant, set “incompressible” to **true**. If there is only one substance, set reducedX also to **true** (the meaning of this flag will be explained below).

In a next step, implementations of model BaseProperties and of all supported functions have to be provided. With the current Modelica language, this is cumbersome, since new classes with different names have to be introduced and then the PartialMedium classes have to be redeclared to the new names. A more convenient Modelica definition could be:

```
 redeclare model BaseProperties
   extends;
   ...
end BaseProperties;
```

This just means that model BaseProperties, which is available due to “extends PartialMedium” is replaced by a model with the same name and all properties defined in PartialMedium.BaseProperties are included via the “extends” statement. This proposed language construct is available as a test implementation in Dymola. At the next Modelica design meeting, a formal decision will be made whether this or something similar will be included into the Modelica language. For the simple air model the redeclaration takes the form:

```
 package SimpleAir
   ...
 redeclare model BaseProperties
   import Modelica.SIunits.
            conversions.*;
   extends(
    p(stateSelect = ..),
    T(stateSelect = ..)
   );
   constant Real R_air = 287.0506;
   constant Real h0    = 274648.7;
 equation
   p = d*R_air*T;
   h = 1005.45*T + h0;
   u = h - p/d;
 end BaseProperties;
   ...
end SimpleAir;
```

The "stateSelect = ..." statements read

```
stateSelect =
      if preferedMediumStates then
         StateSelect.prefer
      else
         StateSelect.default
```

This is the essential definition to decouple balance and medium equations: "preferedMediumStates" is a Boolean parameter defined in PartialMedium. In every device that needs medium properties for balance equations in the form of differential equations, this flag has to be set to **true**. If no derivatives of any of the 5+nX basic thermodynamic variables are needed, this flag has to be set to **false**. Due to the above **if**-expression, the stateSelect attributes of the independent medium variables are set to "**prefer**" if preferedMediumStates = **true**. This in turn means that implicitly equations of the form "pd = **der**(p)" and „Td = **der**(T)" are present and that p and T should be selected as states, if this is possible. This is important, if the property functions, such as u(p,T) are non-linear in the independent variables. If the independent variables would not be selected as states, this would result in non-linear systems of equations for the inversion of the property function.

The balance equations and the medium equations together with the above definition of preferred states define a DAE (= Differential Algebraic Equation system) of index 2. For example, if p and T are used as independent medium variables, this DAE consists of the following equations:

// medium equations

$$d = d(p,T)$$

$$u = u(p,T)$$

$$h = h(p,T)$$

$$pd = \dot{p}$$

$$Td = \dot{T}$$

// balance equations

$$m = d \cdot V$$

$$U = m \cdot u$$

$$\frac{dm}{dt} = \ldots \quad \text{// mass balance}$$

$$\frac{dU}{dt} = \ldots \quad \text{// energy balance}$$

Modelica models often result in higher index DAEs. Dymola solves this problem by using (a) the Pantelides algorithm (Pantelides (1988)) to determine the equations that have to be differentiated and (b) the dummy derivative method (Mattsson and Söderlind (1993), Mattsson et.al. (2000)) to select appropriate states. For the above code fragment, the Pantelides algorithm determines that the equations of m, U and therefore also of d and u need to be differentiated resulting in the following additional equations:

$$\dot{m} = V \cdot \dot{d}$$

$$\dot{U} = \dot{m} \cdot u + m \cdot \dot{u}$$

$$\dot{d} = \frac{\partial d}{\partial p} \cdot \dot{p} + \frac{\partial d}{\partial T} \cdot \dot{T}$$

$$\dot{u} = \frac{\partial u}{\partial p} \cdot \dot{p} + \frac{\partial u}{\partial T} \cdot \dot{T}$$

With the dummy derivative method it is possible to select p and T as states from the original set of potential states (p,T,m,U), especially since p and T have the "prefer" attribute. Using symbolic formula manipulation it is possible to solve the above equations efficiently for $\dot{p}, \dot{T}$.

Note, it is important to set the stateSelect attribute to its default value when preferedMediumStates = **false**. Otherwise, a tool would have to compute the derivative of p and T, although these derivatives are not needed. Worse, in order to compute these derivatives most likely other device equations would have to be differentiated.

After implementation of the BaseProperties model, the optional functions supported by the medium model have to be defined, e.g., a constant dynamic viscosity for the simple air model:

```
package SimpleAir
   ...
redeclare function dynamicViscosity
   input BasePropertiesRecord medium;
   output DynamicViscosity eta;
  algorithm
     eta := 1.82e-5;
end dynamicViscosity;
   ...
end SimpleAir;
```

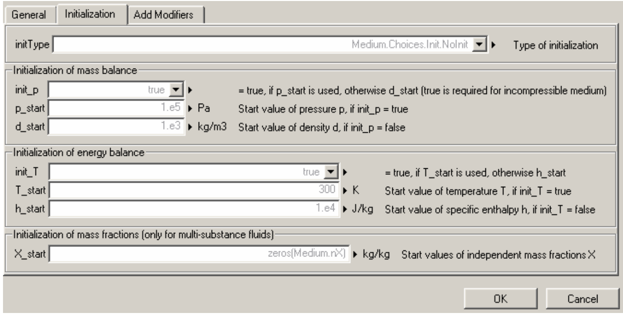
Note, instead of using the short "extends;" as in the BaseProperties model, it is also possible to just repeat the declaration of the function (this is possible with Modelica's type system). For the optional functions, this is a bit longer but seems to be easier to understand for someone looking up the function definition.

The essential part of the medium model is now defined and can be utilized. However, there are additional issues that have to be taken into account, especially for non-linear medium models. This is discussed in the next subsections.

5.3 Initialization

Since variables of the medium are used as states, and the device models using the medium model do (on purpose) not know what independent variables are defined in the medium, initialization has to be defined in the medium model.

For fluid modeling, two types of standard initializations are common: steady state and prescribed initial conditions. A third alternative is additionally supported in the Modelica_Media library: The time scales of the energy- and mass balance related dynamics can be very different for fluid systems and are therefore treated differently in the initialization. A potential state that is determined by the mass balance dynamics (pressure or density) is initialized in steady state i.e., **der**(d)=0 or **der**(p)=0. A potential state that is determined by the energy balance equation (temperature or specific enthalpy) is directly set (e.g. `T = 300.0` or `h = 2.5e6`). This case occurs also when, e.g., initial temperatures are determined by measurements.



initType Medium.Choices.Init.NoInit Type of initialization

Initialization of mass balance

init_p true = true, if p_start is used, otherwise d_start (true is required for incompressible medium)
p_start 1.e5 Pa Start value of pressure p, if init_p = true
d_start 1.e3 kg/m3 Start value of density d, if init_p = false

Initialization of energy balance

init_T true = true, if T_start is used, otherwise h_start
T_start 300 K Start value of temperature T, if init_T = true
h_start 1.e4 J/kg Start value of specific enthalpy h, if init_T = false

Initialization of mass fractions (only for multi-substance fluids)

X_start zeros(Medium.nX) kg/kg Start values of independent mass fractions X

Figure 6 Initialization menu of PartialMedium

In package PartialMedium, several parameters are declared in order to define the initialization. A Dymola screen shot of the "Initialization" menu tab is shown in **Figure 6**. In the lower part, start values for p or d, T or h, and X can be defined. The meaning of a start value, e.g., whether it is a guess value or a definite start value is defined by the first parameter "initType". It is defined with a selection box containing several alternatives (this is implemented as Integer with annotations to specify the content of the selection box, since Dymola does not yet support Modelica enumerations):

- Selection *NoInit* (the default) does nothing, to allow user-specific initialization.
- Selection *InitialStates* means that the independent variables of the medium model should be initialized with start values.
- Selection *SteadyState* sets the time derivatives of the independent medium variables to zero. The start values are interpreted as guess values for the occurring non-linear algebraic equations.
- Selection SteadyMass sets one of the equations der(p) = 0.0 or der(d) = 0, depending whether p or d is an independent variable of the medium model. The start value for p or d is interpreted as a guess value. The start value for T or h is used to initialize the remaining independent variable of the medium model.

In the lower part of the "Initialization" menu, start values can be defined. If the Boolean `init_p = true`, then the start value `p_start` for pressure is used, otherwise the start value `d_start` for density. Correspondingly, if `init_T = true`, the start value `T_start` for temperature is used, otherwise the start value `h_start` for specific enthalpy. Additionally, for multiple substance fluids, start values for mass fractions `X_start` can be defined. Start values that are not needed are used as initial guesses, where appropriate.

While this is not a fully exhaustive list of useful initializations for fluid models, it provides a broad range of practically important cases.

The above parameters are defined in package PartialMedium. An actual implementation must be provided by every medium model. For example, the simple air model, needs the following additions:

```
package SimpleAir
  ...
redeclare model BaseProperties
  import C = Choices.Init;
  protected
    parameter T_start2 =
       if init_T then
          T_start
       else
```

```
            (h_start - h0)/cp_air;
      parameter h_start2 =
         if init_T then
            cp_air*T_start + h0
         else h_start;
      parameter p_start2 =
         if init_p then
            p_start
         else R_air*d_start*T_start2;
      parameter d_start2 =
         if init_p then
            p_start/(R_air*T_start2)
         else d_start)
    public
    extends(
     p(start = p_start2, stateSelect=..,),
     T(start = T_start2, stateSelect=..,),
     d(start = d_start2),
     h(start = h_start2),
     u(start = h_start2 - p_start2/
               d_start2)
    );
    constant Real R_air  =    287.0506;
    constant Real cp_air =   1005.45;
    constant Real h0     = 274648.7;
```

Above is the first part of the initialization. In the extends clause of the BaseProperties model together with the new protected Section, start values for all variables are calculated from the given start values. This requires to evaluate the medium equations with the given start values. In situations with more complex equations, it is often useful to define them with functions and call the functions for start value calculation and in the equation section. The reason to provide consistent start values for all variables is that these variables are potentially iteration variables in non-linear algebraic loops and need therefore reasonable guess values. It is not known beforehand which iteration variable the symbolic translator will select. In the remaining code, the initialization equations and the medium equations are defined:

```
 initial equation
   if preferedMediumStates then
     if initType == C.InitialStates then
       p = p_start2;
       T = T_start2;
     elseif initType==C.SteadyState then
       0 = der(p);
       0 = der(T);
     elseif initType == C.SteadyMass then
       0 = der(p);
       T = T_start2;
     end if;
   end if;
 equation
   p = d*R_air*T;
   h = cp_air*T + h0;
   u = h - p/d;
 end BaseProperties;
   ...
end SimpleAir;
```

Initial equations are only provided if preferedMediumStates = **true**, i.e., if medium variables should be used as states. Depending on parameter initType, the different initialization equations are defined. These equations depend on the independent variables of the medium model.

5.4 Multiple Substance Media

Media that consist of several (non-reacting) substances are both supported from the Modelica_Media and the Modelica_Fluid library. In Modelica_Media essentially the mass fractions X of the substances are used as independent variables to compute the medium properties. Two common approaches are supported by the Modelica_Media library:

- From the n substances, n-1 substances are treated as independent, i.e., n-1 mass fractions are additional independent variables. If needed, the n-th mass fraction is computed from the algebraic equation X_n = 1- sum(X[1:n-1]).
- All n substances are treated as independent during simulation, i.e., n mass fractions are used as independent variables and there are n additional substance mass balance equations. Since the constraint that the mass fractions sum up to one, is not utilized, a slight drift of the mass fractions may occur. Of course, the initial mass fractions have to be defined such that they are summed up to one (this is checked in the PartialMedium package).

In order to not have special cases, the Modelica_Media and Modelcia_Fluid libraries define the constant "nX" of PartialMedium to be the "number of independent" mass fractions. This might be n-1 or n substances of a multiple substance medium. In order to be able to make some checks, such as for initialization, the constant "reducedX" must be defined. If **true**, nX characterizes n-1 substances, if this flag is **false**, nX characterizies n substances.

Note, for single substance media, no mass fraction vector or substance mass flow rate vector is present, because nX = 0 in this case and zero sized vectors are removed in the code generation phase.

6 Medium Models in Modelica_Media

In this Section, some of the more advanced medium models available in the Modelica_Media package are discussed in more detail. All of them are based on the medium interface described in the last Section.

6.1 High Accuracy Water Model IF97

The Modelica_Media library contains a very detailed medium model of water in the liquid, gas and two phase region based on the IF97 standard [6]. It is an adapted and slightly improved version from the ThermoFluid library (Tummescheit and Eborn (2001), Tummescheit (2002)).

High accuracy thermodynamic properties of fluids are modeled with two kinds of multi-parameter, fundamental equations of state:

- An equation for the specific Helmholtz free energy f(ρ,T) or f(v=1/ρ,T)
- An equation for the Gibbs free enthalpy g(p,T)

For numerical reasons the fundamental equations use dimensionless variables which are most often scaled with the critical parameters. The IF97 industrial steam tables uses both equation types and furthermore divides the overall fluid state into 5 regions in order to achieve high accuracy everywhere with a lower number of parameters. In spite of the complexities of the underlying formulation, the user interface for calling the properties is very simple. The medium interface is implemented with utility functions that have a simple interface, e.g.

```
rho = Water.IF97.rho_ph(p,h);
                  //density
T   = WaterIF97.T(p,h);
                 //temperature
s   = WaterIF97.s_ph(p,h);
                 //specific entropy
```

Common sub-expression elimination and nested inlining of function calls ensure that the computationally expensive call to one of the fundamental equations happens only once. A record containing the fundamental derivatives of the equation of states is used by Dymola in the common sub-expression elimination and is thus only computed once. The fundamental derivatives for the free Helmholtz energy f(ρ,T) are:

$$p = \rho^2 f_\rho = \rho^2 \left.\frac{\partial f}{\partial \rho}\right|_T$$

$$s = -f_T$$

$$p_T = \rho^2 f_{T\rho}$$

$$p_\rho = 2\rho f_\rho + \rho^2 f_{\rho\rho}$$

$$c_v = -T f_{TT}$$

Here the short subscript notation is used for partial derivatives, see explanation above. A similar set of fundamental derivatives exists for the Gibbs free enthalpy g(p,T):

$$s = -g_T$$

$$v = g_p$$

$$v_T = g_{pT}$$

$$v_p = g_{pp}$$

$$c_p = -T g_{pp}$$

From these fundamental derivatives, all other partial derivatives of thermodynamic properties with respect to other properties can be computed using thermodynamic determinants, e.g.

$$\left.\frac{\partial \rho}{\partial p}\right|_h = \frac{\rho(c_v \rho + p_T)}{\rho^2 p_\rho c_v + T p_T^2},\ \left.\frac{\partial \rho}{\partial h}\right|_p = \frac{\rho^2 p_T}{\rho^2 p_\rho c_v + T p_T^2}$$

When needed, e.g. for index reduction to change the states to numerically favorable ones, these partial derivatives can be computed with minimal effort from the fundamental derivatives in the property record. In order to add other Helmholtz-or Gibbs-based equations of state to Modelica_Media, only the fundamental derivatives need to be computed, the functions to compute the standard properties are part of the library.

The partial derivatives are used in two situations where the Modelica_Media properties provide unique features for efficiency and model order reduction. For all property calls that may have to be differentiated for index reduction, efficient derivative functions are provided. A very useful model order reduction for large two-phase heat exchangers is to equate the metal mass and boiling water temperatures, e.g. as in the drum Boiler model in [3]. Equating the temperatures leads to an index reduction problem. The algorithm for index reduction needs to compute the time derivative of temperature as a function of the time derivatives of the states. When pressure p and specific enthalpy h are the states, the expansion reads:

$$\frac{dT}{dt} = \left.\frac{\partial T}{\partial p}\right|_h + \left.\frac{\partial T}{\partial h}\right|_p \text{ if in single phase}$$

$$\frac{dT}{dt} = \frac{\partial T_{sat}}{\partial p} \text{ if in the two phase region}$$

These derivatives are automatically computed when needed without user interaction. This allows writing the equations in the most natural form, as demonstrated in [3]. The same algorithmic procedure is used to transform the "natural" form of the mass- and energy balances into equations using the input to the property routines as states.

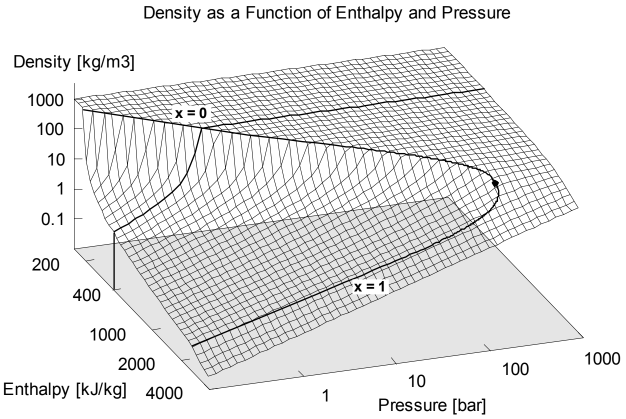


Figure 7: log.-plot of ρ(p,h) for IF97 water

6.2 High Accuracy Ideal Gas Models

Ideal gas properties cover a broad range of interesting engineering applications: air conditioning and climate control, industrial and aerospace gas turbines, combustion processes, automotive engines, fuel cells and many chemical processes. Critically evaluated parameter sets are available for a large number of substances. The coefficients and data used in the Modelica_Media library are from [9]. Care has been taken to enable users to create their own gas mixtures with minimal effort. For most gases, the region of validity is from 200 K to 6000 K, sufficient for most technical applications. The equation of state consists of the well-known ideal gas law $p = \rho \cdot R \cdot T$ with R the specific gas constant, and polynomials for the specific heat capacity $cp(T)$, the specific enthalpy $h(T)$ and the specific entropy $s(T,p)$:

$$cp(T) = R\sum_{i=1}^{7} a_i T^{i-3}$$

$$h(T) = RT\left(-\frac{a_1}{T^2} + a_2\frac{\log(T)}{T} + \sum_{i=3}^{7} a_i \frac{T^{i-3}}{i-2} + \frac{b_1}{T}\right)$$

$$s_0(T) = R\left(-\frac{a_1}{2T^2} - \frac{a_2}{T} + a_3\log(T) + \sum_{i=4}^{7} a_i \frac{T^{i-3}}{i-3} + b_2\right)$$

$$s(T,p) = s_0(T) - R\ln\left(\frac{p}{p_0}\right)$$

The polynomials for $h(T)$ and $s_0(T)$ are derived via integration from the one for $cp(T)$ and contain the integration constants b_1, b_2 that define the reference specific enthalpy and entropy. For entropy differences the reference pressure p_0 is arbitrary, but not for absolute entropies. It is chosen as 1 standard atmosphere (101325 Pa). Depending on the intended use of the properties, users can choose between different reference enthalpies:

1. The enthalpy of formation H_f of the molecule can be included or excluded.
2. The value 0 for the specific enthalpy without H_f can be defined to be at 298.15 K (25 °C) or at 0 K.

For some of the species, transport properties are also available. The form of the equations is:

$$\lg\left(\frac{\nu}{10^k}\right) = A \cdot \lg(T) + \frac{B}{T} + \frac{C}{T^2} + D$$

$$\lg(\lambda) = A_\lambda \cdot \lg(T) + \frac{B_\lambda}{T} + \frac{C_\lambda}{T^2} + D_\lambda$$

$$\eta = \nu \cdot \rho$$

with the kinematic viscosity ν, dynamic viscosity η, thermal conductivity λ and parameters A,B,C,D and k. Note, though, that the thermal conductivity is only the "frozen" thermal conductivity, i.e., not valid for fast reactions.

6.3 Ideal Gas Mixtures

For mixtures of ideal gases, the standard, ideal mixing rules apply:

$$h_{mix}(T) = \sum_{i=1}^{nspecies} h_i(T) x_i$$

$$s_{mix}(T) = \sum_{i=1}^{nspecies} (s_i(T) - R_i \ln(y_i)) x_i - R\ln\left(\frac{p}{p_0}\right),$$

where the x_i are the mass fractions, the R_i are the specific gas constants and the y_i are the molar fractions of the components of the mixture. Most other properties are then computed just as for single species media. Dynamic viscosity and thermal conductivity for mixtures require interaction parameters of a similar functional form as the viscosity itself and are (not yet) implemented.

For mixtures of ideal gases, three usage scenarios can be distinguished:

1. The composition is fixed and is the same throughout the system. This means that a new data record can be computed by preprocessing the component property data that can be treated as a new, single species data record (assuming ideal mixing).
2. The composition is variable, but changes in composition occur only through convection, i.e. slowly.
3. The composition is variable and may change through reactions too, i.e. composition changes are possibly very fast.

Case 1 and 2 above can be handled within a single model with a Boolean switch, case 3 needs to extend from that model because usually a number of additional properties are needed, e.g. the parameters to compute chemical equilibrium reaction constants. Modelica_Media will initially not contain models for reactive flows, but all data is present for users who wish to define such models.

7 Conclusions

Thermodynamic fluid modeling is complex in many ways. This paper has shown a careful structuring of libraries for medium and fluid components in such a way that the same component models can be used with different easily replaceable media. To our knowledge this is the first approach that is able to treat compressible and incompressible fluids in a unified framework. A careful consideration of numerous issues concerning numerical efficiency, model structuring and user friendliness has been presented in this paper:

- Suitable device interfaces
- Principles for handling of reversing, joining and splitting flows
- The governing partial differential equations and their transformation into ODEs
- Pressure loss calculations
- Medium interface design
- Initialization
- Media available in Modelica_Media

Much design effort has been spent on considerations for robust and efficient simulation. The presented framework and libraries have the potential to serve as a powerful base for the development of application-oriented libraries.

Appendix – Energy balance

This appendix contains the derivation of the equivalent but simpler energy balance.

Multiplication of the momentum balance by v gives

$$v\left(\frac{\partial(\rho vA)}{\partial t}+\frac{\partial(\rho v^2 A)}{\partial x}\right)=$$

$$-vA\frac{\partial p}{\partial x}-vF_F-vA\rho g\frac{\partial z}{\partial x}$$

Utilizing the mass balance, this equation can be rewritten as

$$\frac{\partial(\rho(v^2/2)A)}{\partial t}+\frac{\partial(\rho(v^3/2)A)}{\partial x}=$$

$$-vA\frac{\partial p}{\partial x}-vF_F-vA\rho g\frac{\partial z}{\partial x}$$

To show the equivalence, consider the two left hand sides:

$$LH_1=v\frac{\partial(\rho vA)}{\partial t}+v\frac{\partial(\rho v^2 A)}{\partial x}=$$

$$v\left(\frac{\partial v}{\partial t}\rho A+v\frac{\partial(\rho A)}{\partial t}+\frac{\partial v}{\partial x}\rho vA+v\frac{\partial(\rho vA)}{\partial x}\right)=$$

$$v\left(\frac{\partial v}{\partial t}\rho A+\frac{\partial v}{\partial x}\rho vA+v\left(\frac{\partial(\rho A)}{\partial t}+\frac{\partial(\rho vA)}{\partial x}\right)\right)=$$

$$v\left(\frac{\partial v}{\partial t}\rho A+\frac{\partial v}{\partial x}\rho vA\right)$$

$$LH_2=\frac{\partial(\rho(v^2/2)A)}{\partial t}+\frac{\partial(\rho(v^3/2)A)}{\partial x}=$$

$$\frac{\partial(v^2/2)}{\partial t}\rho A+(v^2/2)\frac{\partial(\rho A)}{\partial t}+$$

$$\frac{\partial(v^2/2)}{\partial x}\rho vA+(v^2/2)\frac{\partial(\rho vA)}{\partial x}=$$

$$v\frac{\partial v}{\partial t}\rho A+(v^2/2)\frac{\partial(\rho A)}{\partial t}+$$

$$v\frac{\partial v}{\partial x}\rho vA+(v^2/2)\frac{\partial(\rho vA)}{\partial x}=$$

$$v\left(\frac{\partial v}{\partial t}\rho A+\frac{\partial v}{\partial x}\rho vA\right)$$

i.e. $LH_1=LH_2$.

Subtracting the equation derived above from the energy balance gives

$$\frac{\partial(\rho uA)}{\partial t}+\frac{\partial(\rho v(u+\frac{p}{\rho})A)}{\partial x}=vA\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}(kA\frac{\partial T}{\partial x})$$

Acknowledgements

The design of this library has been a collaborative effort and many have contributed. Many thanks to Mike Tiller for suggesting the package concept and useful discussions and proofreading the paper. Many thanks to Rüdiger Franke for the first realistic tests of the libraries and his feedback, many thanks to Daniel Bouskela, Andreas Idebrant, Gerhart Schmitz, John Batteh, Charles Newman, Jonas Eborn, Sven Erik Mattsson, Hans Olsson and the users of the ThermoFluid library for many useful comments and feedback.

Bibliography

[1] Andersson J.D., Jr. (1995): **Computational Fluid Dynamics – The Basics with Applications**, McGraw-Hill International Editions, ISBN 0 07 001685 2.

[2] Colebrook F. (1939): **Turbulent flow in pipes with particular reference to the transition region between the smooth and rough pipe laws**. J. Inst. Civ. Eng. no. 4, pp. 14-25.

[3] Dymola (2003)*:* **Dymola Users Guide, Version 5.1**. Dynasim AB, http://www.dynasim.se/

[4] Elmqvist, H. (1978): **A Structured Model Language for Large Continous Systems**. PhD-Thesis, Lund Institute of Technology, Lund, Sweden.

[5] Idelchik I.E. (1994): **Handbook of Hydraulic Resistance**. 3rd edition, Begell House, ISBN 0-8493-9908-4

[6] IAPWS (1997); **Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam**. The International Association for the Properties of Water and Steam.

[7] Mattsson S.E.; Söderlind G. (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14 pp. 677-692, 1993.

[8] Mattsson S.E., Olsson H., Elmqvist H. (2000): **Dynamic Selection of States in Dymola**. Modelica Workshop 2000 Proceedings, pp. 61-67, http://www.modelica.org/workshop2000/-proceedings/Mattsson.pdf

[9] McBride B.J., Zehe M.J., and Gordon S. (2002): **NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species**. NASA report TP-2002-211556

[10] Miller D.S. (1990): **Internal flow systems**. 2nd edition. Cranfield:BHRA(Information Services).

[11] Newman C.E., Batteh J.J., Tiller M. (2002): **Spark-Ignited-Engine Cylce Simulation in Modelica**. 2nd International Modelica Conference, Proceedings, pp. 133-142.

[12] Pantelides C. (1988): **The Consistent Initialization of Differential-Algebraic Systems**. SIAM Journal of Scientific and Statistical Computing, pp. 213-231.

[13] Rüdiger Franke (2003): **On-line Optimization of Drum Boiler Startup**. Proceedings of the 3rd International Modelica Conference, Linköping, 2003

[14] Span R. (2000): **Multiparameter Equations of State – An Accurate Source of Thermodynamic Property Data**, Springer-Verlag.

[15] Swamee P.K., Jain A.K. (1976): **Explicit equations for pipe-flow problems**. Proc. ASCE, J.Hydraul. Div., 102 (HY5), pp. 657-664.

[16] Thomas P. (1999): **Simulation of Industrial Processes – For Control Engineers**, Butterworth, Heinemann, ISBN 0 7506 4161 4.

[17] Tummescheit H. (2002): **Design and Implementation of Object-Oriented Libraries using Modelica**, PhD thesis, Department of Automatic Control, Lund Institute of Technology.

[18] Tummescheit H., Eborn J. (2001): **ThermoFluid Modelica Library**. Homepage: http://www.control.lth.se/~hubertus/ThermoFluid/