The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks

Francesco Casella[1], Martin Otter[2], Katrin Proelss[3], Christoph Richter[4], Hubertus Tummescheit[5]

[1]Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italy
[2]German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany
[3]Technical University Hamburg-Harburg, Institute for Technical Thermodynamics, Germany
[4]Technical University Braunschweig, Institute for Thermodynamics, Germany
[5]Modelon AB, Ideon Science Park, Lund, Sweden

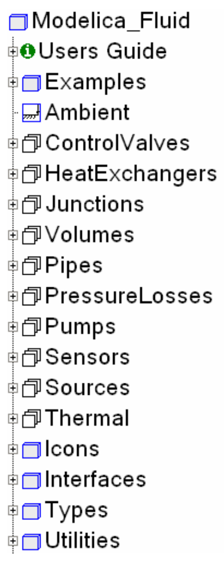
Abstract

The new library Modelica_Fluid is a free Modelica package providing components describing zero- and one-dimensional thermo-fluid components, which can be connected in arbitrary networks. The purpose of the library is to provide standard interfaces for thermo-fluid components, demonstrate how to build such models, and include a growing set of models of common use. The component equations are decoupled from the equations to compute the fluid properties, which are provided by the Modelica.Media library through standard interfaces; incompressible and compressible fluids, single or multiple substances, one- and multiple-phase fluids can be used, where appropriate. Newly introduced features of the Modelica.Media library are briefly reviewed. After extensive testing by interested users, the library will be included in the Modelica standard library as Modelica.Fluid.

1 Introduction

The Modelica_Fluid library provides basic interfaces and components to model thermo-hydraulic systems with zero-dimensional and one-dimensional components. It is not the intention that this library covers all possible application cases, because the modelling assumptions can vary widely. Instead, the goal of the Modelica_Fluid library is to **demonstrate how to implement** components of thermo-hydraulic processes in Modelica, provide **standard connectors** which fit for a wide range of applications, and provide a **reasonable set of components**, which can be used as they are, or can be modified to suit specific user needs. For special applications it is possible to implement libraries with simpler media and components, e.g., the Modelica.Thermal.FluidHeatFlow library [4]. Other domains, such as gas dynamics, would require a more sophisticated setup.

The basic concepts of the Modelica_Fluid library, in particular the fluid connectors and the use of replaceable medium models, were laid out in [2]. Since then, the library design has been refined and tested by several people belonging to the Modelica Association. The structure of the library is now stable – contributions are welcome to increase the number of provided components. The goal is that this library becomes part of the Modelica standard library, after it has been tested by end users on a significant number of different applications and is improved based on the feedback.



A screen shot of the library is shown on the right side. The Examples package contains models that demonstrate various features of the library, as well as some system models, such as a drum boiler [6] and an experimental batch plant [5] model.

A typical (small) example is shown in Figure 1 below: It shows a system where water is pumped from a source by 4 pumps in parallel (fitted with check valves), through a pipe whose outlet is 50 m higher than the source, into a reservoir placed on an 18-m high tower. The users are represented by an equivalent valve, connected to the reservoir. The water controller is a simple on-off controller, acting on the gauge pressure measured at the base of the tower; the output of the controller is the rotational speed of the pumps. A typical simulation is over 2000 s. The pump turns on and off to keep the reservoir level around 2.5 meters, which means

20.5 meters higher than the base of the tower, corresponding to a gauge pressure of 2 bar.

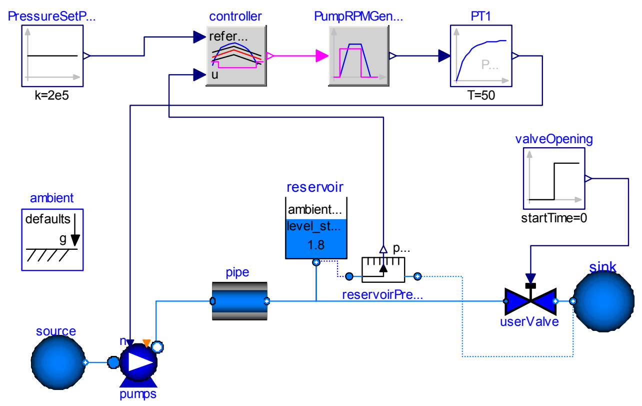


Figure 1: Pumping system for drinking water

2 General design principles

Compared to other engineering modeling fields, such as electrical systems or multibody systems, the task of providing a “standard” library for thermo-fluid systems is much more difficult, due to the much greater variety of modeling assumptions that can be made, depending on the specific application needs. The Modelica_Fluid library tries to strike a balance, providing a sufficiently general framework, which covers a wide range of applications without adding too much overhead to the simplest cases.

The scope of the library includes zero- and one-dimensional models of thermo-hydraulic components, i.e. objects where the flow of one or more fluids must be described, and energy transfer and storage phenomena play a significant role.

The thermo-fluid connectors are designed in order to ensure that mass and energy balances are fulfilled at the connection point, even in presence of flow direction reversal. On the other hand, the momentum balance is fulfilled exactly only when two aligned objects with equal flange diameters are connected; in other cases, the momentum balance at the connecting points is approximated. The exact treatment of momentum balances at the interfaces in those cases would add a significant complexity and overhead to the library, which is unnecessary in most technical thermodynamics applications, where gas dynamics phenomena (wave propagation, high Mach numbers) do not play a significant role. Gas dynamics systems are then outside the scope of the Modelica_Fluid library.

The library models can describe two-phase flows, as long as the flow is homogeneous, i.e., both phases have the same velocity.

The medium models, i.e., the equations to compute all the fluid properties from the independent thermodynamic state variables, are included in the component models as replaceable instances of objects from the Modelica.Media standard library. This allows to use the same component model with different fluids (or with different models of the same fluid) by just replacing the medium model.

3 Fluid Connectors

In this section the design of the fluid connectors is explained. A major design goal was that components can be arbitrarily connected and that the important balance equations are automatically fulfilled when two or more components are connected together at one point as shown in the next figure:

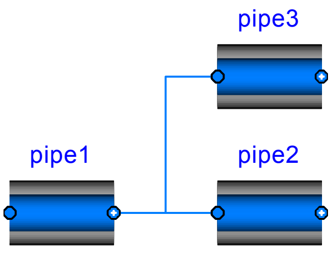


Figure 2: Connected pipes fulfilling the ideal mixing condition at the connection point.

As will be explained below, in such a case the balance equations define ideal mixing, i.e., the connection point has the mixing temperature if the fluids from the three components would be ideally mixed in an infinitely small time period. If more realistic modeling is desired that takes into account dissipation and other mixing losses, an explicit model has to be used in the connection point, e.g., from the Modelica_Fluid.Junctions library. An example is given in the next figure:

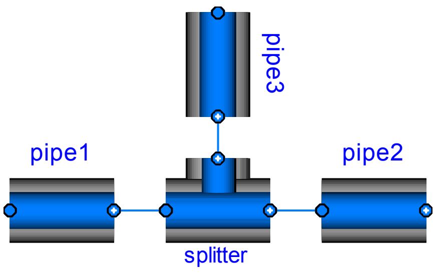


Figure 3: Connected pipes with a splitter junction where the losses are described in the junction model.

For a single substance medium, the connector definition in Modelica_Fluid.Interfaces.FluidPort reduces to

```
connector FluidPort
  replaceable package Medium =
   Modelica.Media.Interfaces.PartialMedium;
  Medium.AbsolutePressure      p;
  flow Medium.MasFlowRate      m_flow;
  Medium.SpecificEnthalpy      h;
  flow Medium.EnthalpyFlowRate H_flow
end FluidPort;
```

The first statement defines the medium flowing through the connector. In package Medium, medium specific types such as "Medium.AbsolutePressure" are defined that contain medium specific values for the min, max and nominal attributes. Furthermore, Medium.MassFlowRate is defined as:

```
type MassFlowRate =
   Modelica.SIunits.MassFlowRate(
      quantity="MassFlowRate." +
                 mediumName, ...);
```

A Modelica translator will check that the quantity and unit attributes of connected interfaces are identical. Therefore, an error occurs, if connected FluidPorts do not have a medium with the same medium name.

The variables in the connector have the following meaning: `p` is the absolute pressure at the connection point, `m_flow` is the mass flow rate from the connection point in to the component, `h` is the specific mixing enthalpy in the connection point and `H_flow` is the enthalpy flow rate from the connection point into the component.

3.1 Balance Equations at Connection Points

Assume that 3 FluidPorts port1, port2, port3, are connected together: Since, m_flow and H_flow are flow variables, a Modelica translator will generate the following connection equations:

```
port1.p = port2.p = port3.p
port1.h = port2.h = port3.h
0 = port1.m_flow + port2.m_flow +
    port3.m_flow
0 = port1.H_flow + port2.H_flow +
    port3.H_flow
```

These are exactly the equations that state ideal mixing for an infinitesimal small control volume in the connection point: The intensive quantities at the ports are identical and the mass balance as well as the energy balance is fulfilled (note that no mass or energy is stored in the infinitesimal volume). The momentum balance is not taken into account, and therefore a connection without an explicit junction model is only valid, if the momentum balance has not much influence or is fulfilled since two ports with the same diameter are connected together.

3.2 Property Propagation over Ports

A connector should have only the minimal number of variables to describe the interface, otherwise there will be connection restrictions in certain cases. Therefore, in the connector no redundant variables are present, e.g., the temperature T is not present because it can be computed from the connector variables pressure `p` and specific enthalpy `h`.

This approach has one drawback: If two components are connected together, then the medium variables on both sides of the connector are identical. However, due to the connector, only the two equations

```
port1.p = port2.p;  port1.h = port2.h;
```

are present. Assume, that `p`, `T` are the independent medium variables and that the medium properties are computed at one side of the connections. This means, the following equations are basically present:

```
port1.h = h(port1.p,port1.T);
port2.h = h(port2.p,port2.T);
port1.p = port2.p;
port1.h = port2.h;
```

These equations can be solved in the following way:

```
port1.h := h(port1.p,port1.T);
port2.p := port1.p;
port2.h := port1.h;
0 = port2.h - h(port2.p,port2.T);
```

The last equation states that `port2.T` is computed by solving a non-linear system of equations. If `port1.h` and `port2.h` are provided as Modelica functions, a Modelica translator, such as Dymola [1], can replace this non-linear system of equations by the equation:

```
port2.T = port1.T;
```

because after alias substitution there are two function calls

```
port1.h := h(port1.p,port1.T);
port1.h := h(port1.p,port2.T);
```

Since the left hand sides of the function calls and the first arguments are the same, the second arguments must also be identical, i.e., port2.T = port1.T. This type of analysis seems to be only possible, if the specific enthalpy is defined as a **function** of the independent medium variables. Due to this requirement, all media in the Modelica.Media library define the specific enthalpy always as a function and therefore by appropriate tool support no unnecessary non-linear system of equation appears and in the generated code, propagation of medium properties over a connector does not lead to an overhead.

3.3 Upstream Discretization

When implementing a fluid component, the difficulty arises that the value of intensive quantities (such as p, T, ρ) shall be accessed from the upstream volume. For example, if the fluid flows from volume A to volume B, then the intensive quantities of volume B have negligible influence on the fluid between the two volumes. On the other hand, if the flow direction is reversed, the intensive quantities of volume A have negligible influence on the fluid between the two volumes. Such a situation is handled with the following code fragment:

```
  import IF = Modelica_Fluid.Interfaces;
  replaceable package Medium =
   Modelica.Media.Interfaces.PartialMedium;
  IF.FluidPort_a port1(redeclare package
                           Medium = Medium);
  IF.FluidPort_b port2(redeclare package
                           Medium = Medium);
equation
  // Handle reverse and zero flow
  port1.H_flow = semiLinear(port1.m_flow,
                         port1.h, port2.h);

  // Energy and mass balance; here:
  port1.H_flow + port2.H_flow = 0;
  port1.m_flow + port2.m_flow = 0;
    ...
```

The enthalpy flow rate in port1 is in principle computed with an if clause:

```
port1.H_flow = port1.m_flow *
                 (if port1.m_flow > 0 then
                     port1.h
                  else
                     port2.h);
```

However, instead of using this if-clause, the corresponding built-in Modelica operator **semiLinear**() is actually used:

```
port1.H_flow = semiLinear(port1.m_flow,
                         port1.h, port2.h);
```

The main reason is that this operator will allow a Modelica translator certain symbolic transformations that lead to a more robust numerical computation (see explanation in the Modelica Specification 2.2).

If the above component is connected between two port volumes (`Modelica_Fluid.Pipes.BaseClasses.PortVolume`), i.e., the independent medium variables in port1 and port2 are states, then port1.h and port2.h are either states (i.e., known quantities in the model) or are computed from states at each integration time step. In such a situation, the above if-clause represented by the "semiLinear" operator is uncritical, because it depends only on known variables and can be directly computed.

If instead, say, pressure loss components are connected, then all port variables are unknown and systems of equations occur. For example, three ports, A.port, B.port, C.port, are connected together. This results in the following equations:

Equations due to
connect(A.port,B.port), **connect**(A.port,C.port):

```
  A.port.p = B.port.p = C.port.p
  A.port.h = B.port.h = C.port.h
  0 = A.port.m_flow + B.port.m_flow +
      C.port.m_flow
  0 = A.port.H_flow + B.port.H_flow +
      C.port.H_flow
```

Equations inside components A,B,C:

```
A.port.H_flow = A.port.m_flow*(
         if A.port.m_flow > 0 then A.port.h
                               else A.h;
B.port.H_flow = B.port.m_flow*(
         if B.port.m_flow > 0 then B.port.h
                               else B.h;
C.port.H_flow = C.port.m_flow*(
         if C.port.m_flow > 0 then C.port.h
                               else C.h;
```

where `A.h`, `B.h`, `C.h`, is the specific enthalpy inside the respective component. All equations together form a linear system of equations to compute the mixing enthalpy `A.port.h` = `B.port.h` = `C.port.h` in the connection point. It has the solution [2]:

```
A.port.h = -( (if A.port.m_flow > 0 then 0
               else A.port.m_flow*A.h)+
              (if B.port.m_flow > 0 then 0
               else B.port.m_flow*B.h)+
              (if C.port.m_flow > 0 then 0
               else C.port.m_flow*C.h) )
          / ( (if A.port.m_flow > 0 then
                     A.port.m_flow else 0)+
              (if B.port.m_flow > 0 then
                     B.port.m_flow else 0)+
             (if C.port.m_flow > 0 then
                    C.port.m_flow else 0) )
```

Therefore, independently of the flow directions in the 3 ports, the mixing enthalpy is always uniquely computed, provided at least one mass flow rate does not vanish (see [2] for details how to handle the case if all mass flow rates vanish). From the mixing enthalpy and the port pressure, all other mixing quantities can be computed, such as mixing temperature.

If two ports A and B are connected together, the resulting system of equations has a solution that is unique also for zero mass flow rates:

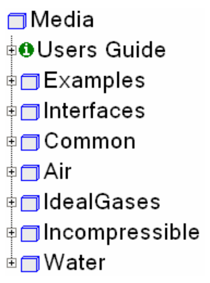
```
  A.port.h = if A.port.m_flow > 0 then B.h
                                   else A.h
  B.port.h = A.port.h
```

In some situations, the user can guarantee that the fluid flows only in one direction. In the Modelica_Fluid library this can be defined in the *Advanced* menu of components by parameter **flowDirection**. Based on this parameter setting, corresponding “min” and “max” attributes are defined for the mass flow rate in a connector, such as:

```
FluidPort_a port_a(m_flow(min = if
                allowFlowReversal then
        -Modelica.Constants.inf else 0)
```

When port_a.m_flow is referenced in a semiLinear() operator, the tool can deduce that only one branch of the if-clause can appear and can utilize only this branch for the further symbolic processing. As a result, if-clauses that define the reversing flow are removed.

4 Medium models



Modelica_Fluid uses the free library Modelica.Media that was developed to provide a standardized interface to media models and a large number of ready-to-use media models based on that interface. The basic concept of Modelica.Media is described in [2]. It was included in the Modelica Standard Library in version 2.2. The library has been continuously improved to fit the requirements of Modelica_Fluid. The picture on the left shows the structure of Modelica.Media. Modelica.Media allows for a decoupling of the formulation of the balance equations within a Modelica_Fluid component model and the definition of the medium. Different interfaces are provided in Media.Interfaces that are used as base classes for the implementation of different medium models of different nature, e.g., ideal gases, real gases, two-phase mediums. For every medium a record called ThermodynamicState is implemented that contains the minimum set of variables required to describe the state of the medium. The thermodynamic state record for a pure component ideal gas is

```
record ThermodynamicState
  SI.AbsolutePressure p;
  SI.Temperature      T;
end ThermodynamicState;
```

The thermodynamic state record can be used to compute all other fluid properties except for the saturation properties which will be explained later. The functions to compute additional fluid properties are all contained within package Media.Interfaces. A function without an underscore in its name assumes the thermodynamic state record as an input. The function `specificEnthalpy()` for example will compute the specific enthalpy from the thermodynamic state.

The following code fragment demonstrates how the thermodynamic state record could be used in a simple component model to compute all required fluid properties:

```
replaceable package Medium =
  Modelica.Medium.Interfaces.PartialMedium;
Medium.ThermodynamicState state;
Medium.SpecificEnthalpy h;
    ...
state = Medium.setState_pT(1e5, 273.15);
h = Medium.specificEnthalpy(state);
```

The function `setState_pT()` will return the state for the given input variables pressure (p) and temperature (T) independently from the actual entries in the thermodynamic state record. For example, if the medium state is p and h and `setState_pT(..)` is called, for most media a non-linear equation in one unknown will be solved to compute h (this computation is performed reliably and efficiently). The second part of the function name following the underscore indicates the required input variables which is the standard for all function names within Modelica.Media. The more general function to compute the state would be `setState_pTX()` which also requires the nX mass fractions X[nX] for a multiple substance medium as input. Using the thermodynamic state record in models is a more function-based approach to medium modeling and is used in static components, e.g., pressure loss models or the heat transfer to the wall of a pipe.

Modelica.Media also offers an object-oriented approach that uses the model `BaseProperties` defined for each medium interface. This approach is more suitable for dynamic component models, e.g., a volume or a tank, than the function-based approach. The provided base property model can be extended by the user to best meet the specific requirements. The purpose of using the thermodynamic state model in the function based and in the object oriented approach is to be able to write models that are independent of the input variables to the fluid property model. The state selection mechanism described in [2] makes it possible to obtain numerically efficient models for different fluids with the same component models. The basic idea is sketched at hand of the following implementation of a port volume:

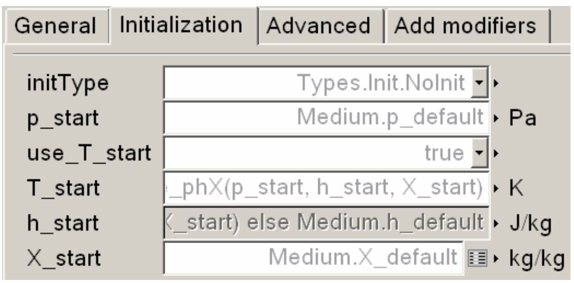
```
  replaceable package Medium =
   Modelica.Media.Interfaces.PartialMedium;
  Modelica_Fluid.Interfaces.FluidPort_a
   port(redeclare package Medium = Medium);
  Medium.BaseProperties medium (
            preferredMediumStates = true);
equation
  medium.p = port.p;
  medium.h = port.h;
         M = V*medium.d;
         U = M*medium.u;
    der(M) = port.m_flow; // mass balance
    der(U) = port.H_flow; // energy bal.
```

In a port volume it is desired that the independent medium variables are used as states (e.g., p,T or p,h depending on the medium). The BaseProperties instance medium contains the basic medium equations. If parameter *preferredMediumStates* is set to **true**, then attribute StateSelect.prefer is set to the independent medium variables and therefore the tool will use these variables as states for the mass and energy balance, if this is possible. This means, that the port volume equations can be implemented without knowledge about the independent medium variables.

`Modelica.Media` requires the implementation of medium models in Modelica. This approach allows the solver to use as much analytical information about the medium models as possible when manipulating the system of equations. However, it is often also very desirable to use existing fluid property libraries written in C or in FORTRAN. A new interface to an external medium library has been developed for `Modelica.Media` that supports external medium libraries. This new interface is currently included in the developer version of `Modelica.Media` and will be tested thoroughly before including it in the Modelica Standard Library.

5 Initialization

Every fluid component with states has a menu "Initialization". A screen shot of this menu of model Modelica_Fluid.Volumes.MixingVolume is shown in the next figure:



General | Initialization | Advanced | Add modifiers

initType	Types.Init.NoInit	
p_start	Medium.p_default	Pa
use_T_start	true	
T_start	_phX(p_start, h_start, X_start)	K
h_start	(_start) else Medium.h_default	J/kg
X_start	Medium.X_default	kg/kg

Parameter initType defines the type of the initialization and has the following options:

- initType == InitialValues:
  Initial values of p,X and of T or h are defined.
- initType == SteadyState:
  The derivatives of the states are set to zero during initialization. Since usually non-linear systems of equations occur, guess values for the states are defined for p, X and for T or h.
- initType == SteadyStateHydraulic:
  The pressure derivatives are set to zero during initialization, but the thermal states (T or h) are initialized with a start value. Therefore, a guess value for p and initial values for X and for T or h are defined.

Depending on the selected option, a value such as "p_start" is interpreted from the component as either being an **initial value** (i.e. introducing an initial equation p = p_start) or a **guess value** (i.e. setting the start value of p to p_start with fixed = **false**).

For every medium either T or h can be defined as start value. Assume that T_start is selected as value to be provided (either initial or guess value). Depending on the situation, a tool might use h as iteration variable for a non-linear system of equations, e.g., because h is the independent medium variable. Then, the setting of T_start would have no effect. For this reason, modifiers are defined in the initialization menu, e.g. for h_start:

```
parameter Medium.SpecificEnthalpy h_start=
   if use_T_start then
      Medium.specificEnthalpy_pTX(
                p_start, T_start, X_start)
   else Medium.h_default;
```

If use_T_start is true, the menu for h_start is disabled, i.e., the user cannot input a value and therefore function specificEnthalpy_pTX(..) is called to compute the start value of the specific enthalpy based on p_start and T_start. If use_T_start = **false**, the user can provide a modifier with a new value that overwrites the if-clause in the modifier. Otherwise the default value of h for this medium is used as initial value.

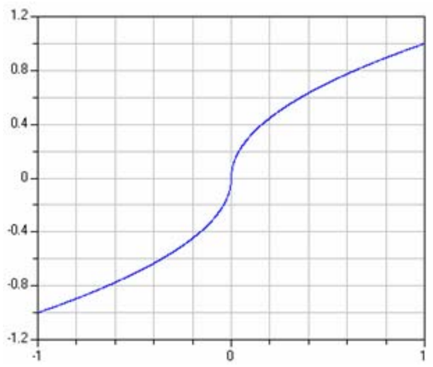
To summarize, the medium is always initialized with a consistent set of variables p, T, h, X where either T or h is computed from the other 3 variables with the corresponding medium function.

6 Regularizing characteristics

Pressure drop equations and other fluid characteristics are usually computed by **semi-empirical** equations. Unfortunately, the developers of semi-empirical equations nearly never take into account that the equation might be used in a simulation program. As a consequence, these semi-empirical equations can nearly never be used blindly but must be slightly modified or adapted in order that obvious simulation problems are avoided. For example, turbulent flow in a pipe might be described by the following type of equation:

```
    y = if x >= 0 then  sqrt(k1*x)
                  else -sqrt(k2*abs(x));
```

A plot of this characteristic is shown in the next figure:

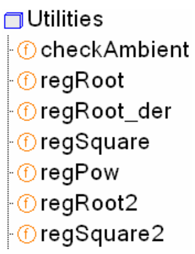


The difficulty with this function is that the derivative at x=0 is infinity. The actual physical characteristic doesn't show this singularity. E.g., for pipe flow, the flow becomes laminar for small velocities and therefore around zero the **sqrt**() function is replaced by a linear function. Since the laminar region is usually of not much practical interest, the above approximation is used.

The direct implementation above does not work in Modelica, because an event is generated when `x < 0` changes sign. In order to detect this event, event iteration takes place. During the event iteration, the active if-branch is not changed. For example, assume that x is positive (= "**else**" branch) and shall become negative. During the event iteration `x` is slightly negative and the **else** branch, i.e., `sqrt(x)`, is evaluated. Since this result in an imaginary number, an error occurs. It would be possible to fix this, by using the `noEvent()` operator to explicitly switch off an event:

```
y = noEvent( if x<0 then  sqrt(k1*x)
                   else -sqrt(k2*abs(x)));
```

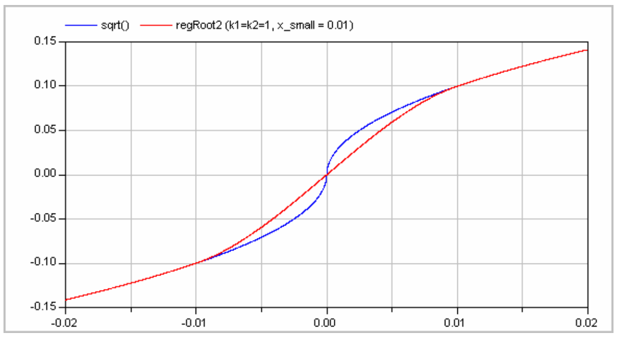
Still, it is highly likely that good integrators will not work well around `x=0`, because they will recognize that the derivative changes very sharply and will reduce the step size drastically.


Utilities
checkAmbient
regRoot
regRoot_der
regSquare
regPow
regRoot2
regSquare2

In Modelica_Fluid.Utilities several utility functions are provided to regularize such types of equations (see screen shot on right side). For example, regRoot2(..) replaces the function above by two polynomials of third order around zero, so that the overall function is continuous, is strict monotonically increasing and has a continuous first derivative everywhere. Additionally, either the second derivatives of the two polynomials at zero are identical (= default) or a user defined first derivative at zero can be provided, to, e.g., correctly describe the laminar region around zero. In the first case, the equation above is replaced by:

```
y = regRoot2(x, x_small, k1, k2);
```

where x_small defines the region of the newly introduced two polynomials around x = 0. The result of applying this function is shown in the next figure.



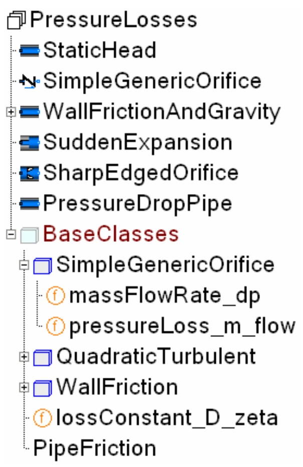
The "blue" curve is the exact characteristic according to the equation above, where as the "red" curve is the regularized approximation of regRoot2(..) that has much better numerical properties.

7 Selected Components

In the previous sections, the features have been described that are needed in order that component models can be implemented. In this section some of the provided component models will be shortly sketched.

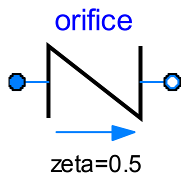
7.1 Pressure Losses

Package PressureLosses contains models and functions providing pressure loss correlations. All models in this library have the property that no mass and no energy is stored in the component. Therefore, none of the models has a state. The basic correlations are **models** that are implemented with **functions** of sublibrary PressureLosses.BaseClasses. These functions might also be directly called (e.g. in an implementation of another component, such as the distributed pipe).


PressureLosses
StaticHead
SimpleGenericOrifice
WallFrictionAndGravity
SuddenExpansion
SharpEdgedOrifice
PressureDropPipe
BaseClasses
SimpleGenericOrifice
massFlowRate_dp
pressureLoss_m_flow
QuadraticTurbulent
WallFriction
lossConstant_D_zeta
PipeFriction

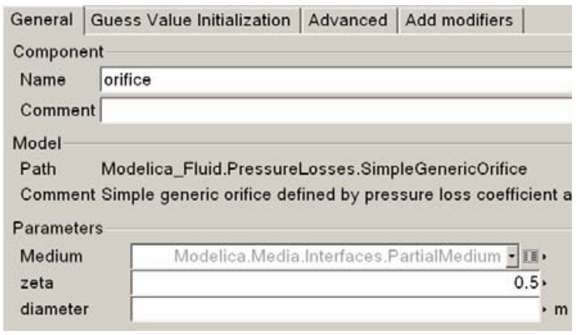
All functions are continuous and have a finite, non-zero, smooth, first derivative. The functions are all guaranteed to be strict monotonically increasing. The mentioned properties guarantee that a unique inverse of every function exists. In fact, for all correlations a function is provided in the form m_flow = f(p) and also its inverse, p = g(m_flow) is given. A similar naming convention as in the Media library is used, e.g. massFlowRate_dp(..) means that the functions compute the mass flow rate and that the input argument is dp (the pressure difference between two ports). Most functions consist of one statement, so that, e.g., Dymola inlines the function and therefore no call overhead is present.

The pressure loss correlation "SimpleGenericOrifice" defines a standard quadratic correlation of the form:



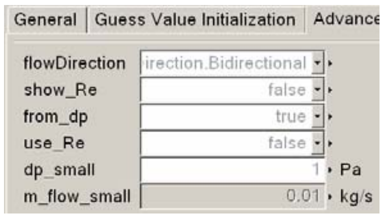
$$\Delta p = \frac{1}{2} \cdot \zeta \cdot \rho \cdot v |v|$$

where Δp is the pressure difference between two ports, v is the fluid velocity (that can be computed from the mass flow rate, density and pipe area) and ζ is the constant pressure loss coefficient, for the fluid flow from port_a to port_b that can be, e.g., deduced from some of the standard books like Idelchick [3]. Screen shots of the parameter menu are shown in the next two figures:



Component	
Name	orifice
Comment	
Model	
Path	Modelica_Fluid.PressureLosses.SimpleGenericOrifice
Comment	Simple generic orifice defined by pressure loss coefficient a
Parameters	
Medium	Modelica.Media.Interfaces.PartialMedium
zeta	0.5
diameter	m

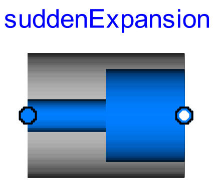
Basically, the medium, the correlation factor and the diameter has to be defined at which ζ is defined. The "Advanced" menu is the same for all components of the PressureLosses package and defines how the computation of the correlation is performed:



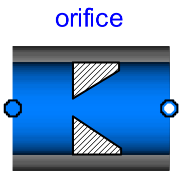
flowDirection	irection.Bidirectional	
show_Re	false	
from_dp	true	
use_Re	false	
dp_small	1	Pa
m_flow_small	0.01	kg/s

If from_dp is true, the mass flow rate is computed from the pressure drop, otherwise the computation is reversed. The "flowDirection" defines whether reversal flow shall be taken into account. "use_Re" defines the laminar region by the Reynolds number (e.g. Re < 2000 for smooth wall friction), otherwise it is defined approximately by a small pressure drop or a small mass flow rate depending on the selected computation direction. Finally, if show_Re = true, the Reynolds-Number is computed in order to utilize it, e.g., in a plot. By default the computations with the Reynolds number are not performed, since a medium model may not provided a function to compute the viscosity.

Model "suddenExpansion" defines a sudden expansion of a pipe and computes the correlation factors for the two flow directions from the two pipe diameters according to Idelchick [2].



In the same way "orifice" defines a sharp edged orifice where the correlation factors for the two flow directions depends, e.g., on the opening angle of the orifice [2].



Model "StaticHead" models only the pressure drop due to gravity.

Finally, model "WallFrictionAndGravity" models wall friction and also takes into account gravity. The implementation is based on [2,3]. The user can select either the different regions (only laminar, only quadratic turbulent, laminar + quadratic turbulent) or the detailed characteristic. The latter one is shown in the next figure [2,3].

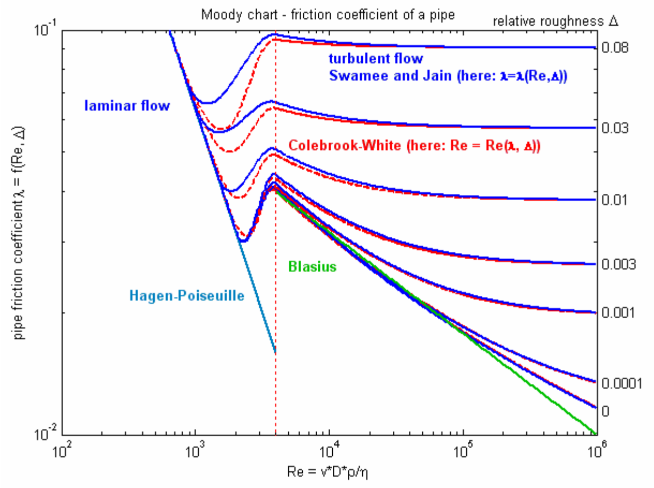
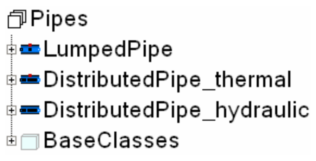


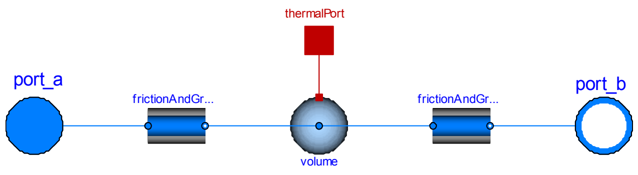
Figure 1. Moody Chart: lg(λ) = f (lg(Re), Δ), ζ= λL/D

7.2 Pipes

Different pipe models are defined in package Pipes, as shown in the screen shot at the right. LumpedPipe is a simple pipe model consisting of one volume and two pressure loss correlations for the wall friction, as well as a heat transfer port to describe the heat transfer through the wall. The model is especially useful for demonstration purposes because it is just built from basic components:



Pipes
- LumpedPipe
- DistributedPipe_thermal
- DistributedPipe_hydraulic
- BaseClasses



The other two pipe models are discretized pipes consisting of n volumes. More details are given in the next subsection.

7.3 Heat Exchanger

A basic heat exchanger model can be found under Components.HeatExchangers.BasicHX. It demonstrates the usage of several models from the Fluid library and the interfaces provided to adapt them to fit personal needs. The heat exchanger is composed of two pipe flow models and one wall element as shown in figure 4. The wall determines a co- or counterflow orientation of the two medium flows. It also adds the major thermal capacity to the set. Heat conduction is assumed to be one-dimensional, perpendicular to both fluid flows.

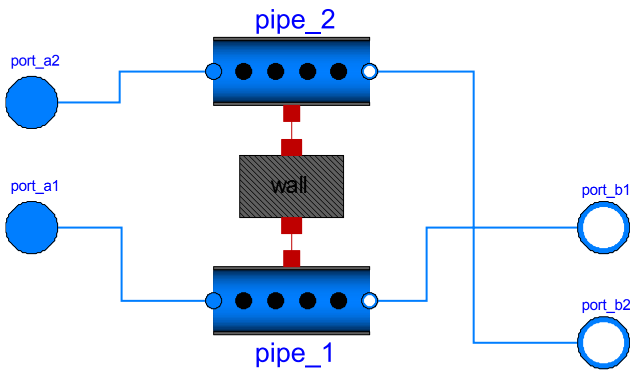


Figure 4: Heat exchanger component

On both fluid sides medium packages from the Modelica.Media library can be chosen. An instance of the respective `BaseProperties` model as described in section 4 is automatically included in each of the two distributed flow models from the component package `Pipes`. They follow an upwind discretization scheme, the number of segments being the same for both pipes and the wall. Dynamic energy and mass balances interlace on a staggered grid with static momentum balances for each control volume. Two half momentum balances on each end make the component fully symmetric. The port interface corresponds to the general design principle outlined in section 3 and allows for flow reversal. A uniform cross sectional area is assumed along the entire flow path.

Empirical heat transfer and pressure drop correlations allow us to reduce 3D fluid flow problems to one dimension. They largely depend on the specific application, thus have to be replaceable in a model in order to provide the required flexibility, but at the same time need to be known in the lowest hierarchical level of a system, the governing balance equations.

The distributed pipe model contains a replaceable heat object that determines the relationship between the thermal port properties, heat flow and temperature, and the bulk flow, namely the medium temperature and the sensible heat term in the energy balance. The library currently only provides the simplest model possible to describe a sensible heat transfer, by means of a constant heat transfer coefficient. But an implemention of e.g. Nusselt correlations from the literature is easily done by inheriting from the base model `Pipes.BaseClasses.HeatTransfer.PartialPipeHeatTransfer`. Besides geometrical parameters, such as the hydraulic diameter and cross sectional flow area the heat object also "knows" mass flow rate and the `medium.state` record (see section 4) of the fluid flow, which makes it possible to compute required transport properties by function call if and only if needed in the respective correlation. For further information concerning the models mentioned here the reader may be referred to the online documentation of the library.

Figure ? shows the results of an example model in the library. One of the two fluid flows in the heat exchanger changes its direction midway, and because it is fed from a colder source changes the direction of heat flow.

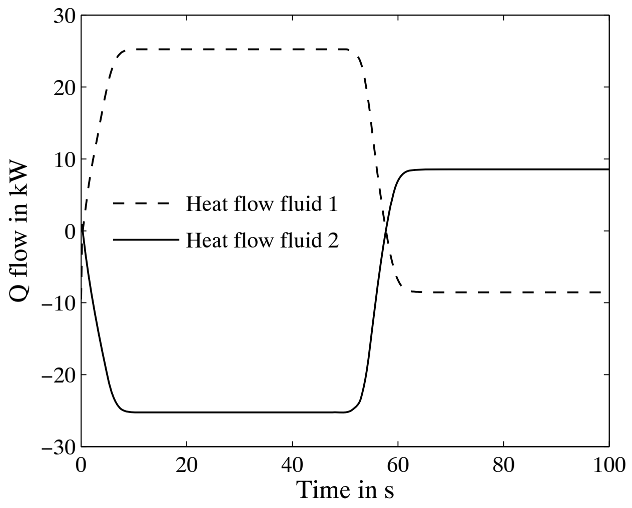


Figure 5: Heat flow rates in both heat exchanger fluids (water) while one of them changes direction.

8 Conclusions

The 1.0 Beta 1 version of the Modelica_Fluid library described in this article is in a rather stable stage and the most important basic problems have been resolved. Especially, it was possible to reach the following quite ambitious goals:

(a) The component equations are independent from the medium equations (especially, a component can be used for media that have different sets of independent variables, such as T, pT, or p,h, or p,T,X or T,X etc.). This has the big advantage that pump, pipe, valve models etc., can be implemented just once and utilized for quite different media. Of course, there are limits, e.g., one and two phase flow is always differently described in a component. On the other hand, all components of the Fluid library support incompressible and compressible as well as one and multiple substance media.

(b) Components can be arbitrarily connected together. Also models such as a pipe can be flipped. The Modelica connection semantics generates ideal mixing equations so that the mass and energy balance is fulfilled. If this is not desired, junction models have to be used. This is especially the case when the momentum balance in a junction cannot be neglected. There are still some unresolved issues, e.g., the Pipes.DistributedPipe model is discretized in such a form that at the two ends of a pipe momentum balances are present (and not mass and energy balances of a volume). When connecting pipes of this form directly together (without using a port volume in the connection point), non-linear systems of equations appear.

The goal is to continuously improve the Modelica_Fluid library, especially to include more component models. Contributions from users of the library are welcome. The actual version of the library can be downloaded from
http://www.modelica.org/library/

9 Acknowledgments

The development of the Modelica_Fluid library started in year 2002 and many have contributed:

The Fluid library development was organized in 2002-2004 by Martin Otter and since 2004 it is organized by Francesco Casella. The essential basic design of the Fluid library, especially component interfaces, handling of reversing flow with the semiLinear() operator, property propagation is from Hilding Elmqvist. Besides the authors, the following people contributed to the fluid component models, examples, symbolic algorithms and the further design of the library (alphabetical list): John Batteh, Jonas Eborn, Rüdiger Franke, Anton Haumer, Henning Knigge, Chuck Newman, Hans Olsson, Katja Poschlad, Manuel Remelhe, Sven Erik Mattsson, Mike Tiller, Allan Watson.

The Modelica.Media library development is organized by Hubertus Tummescheit. For the long list of contributors, see
Modelica.Media.UsersGuide.Contact.

10 References

[1] Dynasim (2006). *Dymola Version 6.0*. Dynasim AB, Lund, Sweden. Homepage: http://www.dynasim.se/.

[2] Elmqvist, H., Tummescheit H., and Otter M.(2003). *Object-Oriented Modeling of Thermo-Fluid Systems*. Proceedings of 3rd Int. Modelica Conference, Linköping, Sweden, ed. P. Fritzson, pp. 269-286. http://www.modelica.org/Conference2003/papers/h40_Elmqvist_fluid.pdf

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

[4] Kral A., Haumer A. Plainer M. (2005): *Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library*. 4th int. Modelica Conference, Hamburg-Harburg. http://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3b1.pdf

[5] Poschlad K., Remelhe M.A.P., and Otter M. (2006): *Modeling of an Experimental Batch Plant with Modelica*. 5th int. Modelica Conference, Vienna.

[6] Rüdiger Franke (2003): *On-line Optimization of Drum Boiler Startup*. Proceedings of the 3rd Int. Modelica Conference, Linköping, 2003. http://www.modelica.org/events/Conference2003/papers/h29_Franke.pdf