Design choices for thermofluid flow components and systems that are exported as Functional Mockup Units

Michael Wetter[1] Marcus Fuchs[2] Thierry S. Nouidui[1]

[1]Lawrence Berkeley National Laboratory, Energy Technologies Area, Building Technology and Urban Systems Division, Simulation Research Group, Berkeley CA, USA, {mwetter,tsnouidui}@lbl.gov
[2]RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany, mfuchs@eonerc.rwth-aachen.de

Abstract

This paper discusses design decisions for exporting Modelica thermofluid flow components as Functional Mockup Units. The purpose is to provide guidelines that will allow building energy simulation programs and HVAC equipment manufacturers to effectively use FMUs for modeling of HVAC components and systems.

We provide an analysis for direct input-output dependencies of such components and discuss how these dependencies can lead to algebraic loops that are formed when connecting thermofluid flow components. Based on this analysis, we provide recommendations that increase the computing efficiency of such components and systems that are formed by connecting multiple components. We explain what code optimizations are lost when providing thermofluid flow components as FMUs rather than Modelica code. We present an implementation of a package for FMU export of such components, explain the rationale for selecting the connector variables of the FMUs and finally provide computing benchmarks for different design choices. It turns out that selecting temperature rather than specific enthalpy as input and output signals does not lead to a measurable increase in computing time, but selecting nine small FMUs rather than a large FMU increases computing time by 70%.

Keywords: FMI, Modelica, thermofluid flow

1 Introduction

The Functional Mockup Interface Standard (Modelica Association, 2014) is an open standard that has been developed to export models or whole simulators from one simulation software and import them into another simulation software to perform a coupled simulation of time dependent systems. It enables interoperability among simulation software by standardizing (i) an application programming interface and its semantics, (ii) an xml schema that describes the model structure and capabilities, and (iii) the structure of a zip file that is used to package the model, its resources and documentation.

This type of simulation software interoperability is interesting for various use cases in building energy simulation. First, it allows building energy simulation programs, for which it currently is difficult for users to add new models, to add an interface that allow users to insert own component models that may be written in and exported by a variety of simulation software that support the FMI standard (see https://www.fmi-standard.org/tools for a list). As a point in case, EnergyPlus currently undergoes a prototype redesign in which HVAC simulation will be based on FMUs (Wetter et al., 2015). Second, the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) is currently developing Standard 205 that standardizes the representation of HVAC equipment performance data for building energy simulation.[1] As the built-in control and staging algorithms of such equipment affects the performance, participants of the standards committee expressed the need for sharing models as executable code rather than simple performance maps. Here, FMU may be a solution for such model representation. Third, Swegon AB, an international HVAC equipment manufacturer, expressed the need for receiving from their suppliers component models to allow them to optimize the integration of these components into their products. Swegon also is interested in providing equipment models as FMUs to energy simulation programs.

For these use cases, FMI is an interesting technology as it is an open standard that has been designed for the exchange of such models. However, various design questions have to be answered for its effective use, namely: (i) Are both versions of the standard, FMI for model-exchange and FMI for co-simulation, applicable? (ii) If an FMU represents an individual equipment, how would a system simulation program have to execute this component if used as part of an whole HVAC simulation? (iii) What recommendations should one follow to allow an efficient simulation of FMUs if part of an HVAC system simulation? (iv) What code optimization is lost if FMUs are used rather than Modelica, the latter allowing

[1] See http://spc205.ashraepcs.org/.

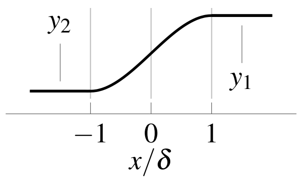

Figure 1. Plot of the function $y = \mathscr{R}(x, y_1, y_2, \delta)$.

symbolic processing prior to code generation. (v) What variables should the parameters, inputs and outputs of an FMU have? Would specific enthalpy as is used in the `Modelica.Fluid` library be a good choice?

This paper addresses the above questions. It is structured as follows: Section 2 states assumptions and introduces notation. In Section 3, we discuss the applicability of the two standards. In Section 4, we provide an analysis of execution sequences and provide recommendations for efficient model implementation. In Section 5, we discuss the design of connector variables. Section 6 explains code optimizations that are no longer possible if FMUs rather than Modelica are used. Section 7 discusses the implementation of FMU export for thermofluid flow components in a development version of the Modelica Buildings library (Wetter et al., 2014), and Section 8 shows numerical benchmarks for different implementations.

2 Terminology and assumptions

2.1 Conventions

1. If a component has two acausal fluid ports, then they are denoted by subscript a and b, respectively, where a is the port into which mass flows under the design flow direction.
2. Consider a model that has an input $u \in \mathbb{R}$, an output $y \in \mathbb{R}^2$ and a parameter $p \in \mathbb{R}$. Suppose $y_1 = pu$ and $dy_2/dt = u$. We say that there is a *direct dependency* between u and y_1 as the value of u needs to be known to produce the output y_1. In contrast, y_2 does not directly depend on u as it can be produced without knowing the current value of u.

2.2 Notation

To enable robust iterative computation of a numerical approximation to the solution of differential and algebraic systems of equations, the Modelica Specification defines special functions to handle systems in which the flow reverses its direction, and the Modelica Standard Library 3.2 implements regularization functions (Franke et al., 2009; Modelica, 2010). This section explains these functions and the nomenclature that we will use for these functions.

The regularization function `Modelica.Fluid.Utilities.regStep(x,y1,y2,x_small)` approximates

$$y = \begin{cases} y_1, & \text{if } x > 0, \\ y_2, & \text{otherwise,} \end{cases} \tag{1}$$

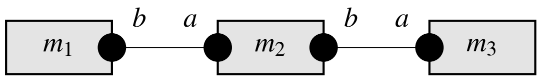

Figure 2. Connection diagram of three models that is used to explain the concept of stream variables.

by the once continuously differentiable function that is shown in Figure 1. In our discussions, we will denote this function by $y = \mathscr{R}(x, y_1, y_2, \delta)$, with $\delta > 0$. The function is defined as

$$\mathscr{R}(x, y_1, y_2, \delta) = \begin{cases} y_1, & \text{if } x > \delta, \\ y_2, & \text{if } x < -\delta, \\ r(x, y_1, y_2, \delta), & \text{otherwise.} \end{cases} \tag{2}$$

where $r(\cdot, \cdot, \cdot, \cdot)$ is

$$r(x, y_1, y_2, \delta) = \frac{x}{\delta}\left(\left(\frac{x}{\delta}\right)^2 - 3\right)\frac{y_2 - y_1}{4} + \frac{y_1 + y_2}{2}. \tag{3}$$

Note that some models use `Modelica.Media.Air.MoistAir.Utilities.spliceFunction()` rather than `Modelica.Fluid.Utilities.regStep()`. While these functions are different, their input-output dependency is identical. We will therefore always use the notation $\mathscr{R}(\cdot, \cdot, \cdot, \cdot)$ as our discussion is identical for both implementations. [2]

To describe in a numerically reliable way the bi-directional transport of specific quantities that are carried by mass flow rate, such as enthalpy, Modelica 3.2 provides the `inStream()` function. Let `m1`, `m2` and `m3` be models, and let `a` and `b` be fluid ports that are connected as shown in Figure 2. Let `h_outflow` be the specific enthalpy in the connection point if mass leaves the component (regardless of the current flow direction). For the configuration shown in Figure 2, the `inStream()` function satisfies

```
inStream(m2.a.h_outflow) = m1.b.h_outflow;
inStream(m2.b.h_outflow) = m3.a.h_outflow;
```

In our discussions, we will use the notation $\imath(h_a)$ to denote the value of `inStream(a.h_outflow)`.

2.3 Assumptions

We will make the following assumptions:

1. All components conserve mass, e.g., $\sum_i \dot{m}_i + \Delta\dot{m} = 0$ where the sum is over all ports and $\Delta\dot{m}$ is the moisture added or removed by a humidifier or a cooling coil.
2. Each component has as inputs the mass flow rate $\dot{m}_a$, the pressure p_a, the temperature $T_{a,i}$ (or specific enthalpy $h_{a,i}$) of the medium that flows into port a if $\dot{m}_a \geq 0$, and the temperature $T_{b,i}$ (or specific enthalpy $h_{b,i}$) of the medium that flows into port b if $\dot{m}_a < 0$.

[2]In a benchmark for the Annex 60 model library (see `https://github.com/iea-annex60/modelica-annex60/issues/300`) we measured that the function `regStep()` is on average about 8% faster than `spliceFunction()`. Therefore, work is in progress to update the `Annex60` and `Buildings` libraries accordingly.

3. Each component has as outputs the mass flow rate $\dot{m}_b$, the pressure p_b, the temperature $T_{b,o}$ (or specific enthalpy $h_{b,o}$) of the medium that flows out of port b if $\dot{m}_a \geq 0$, and the temperature $T_{a,o}$ (or specific enthalpy $h_{a,o}$) of the medium that flows out of port a if $\dot{m}_a < 0$.
4. We assume the following direct dependencies: The outlet temperatures are $T_{b,o} = f(T_{a,i}, \dot{m}_a)$ and, similarly, $T_{a,o} = g(T_{b,i}, \dot{m}_a)$ for some functions $f, g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.
5. The pressure drop of flow resistances is assumed to be a function of the mass flow rate rather than volume flow rate. The reason for this assumption will become clear in Section 4.

Hence, to simplify the discussion, we typically say that input and output to a component are temperature T. Clearly, models that treat moist air also have water vapor mass fraction X_w as input and output. Except for the discussion of dehumidifying or humidifying components, we do not specifically mention water vapor mass fraction as other components conserve mass. Furthermore the discussion also holds if one were to use specific enthalpy rather than temperature as input and output variables.

3 FMI Standards

FMI 2.0 defines two standards: FMI for model-exchange (FMI-ME) and FMI for co-simulation (FMI-CS): In FMI-ME, the host simulator is responsible for the numerical integration of the model equations, whereas in FMI-CS, the FMU implements its own mechanism for advancing the values of its state variables. FMI-CS provides no mechanism for an FMU to output an instantaneous reaction to a changed input value.[3] Hence, FMI-CS cannot be used for steady-state component models of HVAC equipment. However, FMI-ME is applicable. Specifically, FMI-ME allows to set inputs by calling `fmi2SetReal` followed directly by `fmi2GetReal` to obtain outputs. Furthermore, the standard says that `fmi2SetReal` "re-initializes caching of variables that depend on these variables [being set]". Hence, `fmi2SetReal` causes the equations to be evaluated. Therefore, we restrict this discussion to FMI-ME.

4 Direct input-output dependencies of thermofluid flow components and systems

The purpose of this section is to provide guidance to users and developers who connect multiple thermofluid flow component models so they understand when algebraic loops are performed, and how such algebraic loops can be avoided. While in general the existence of algebraic loops can readily be obtained from the translation information of Modelica tools, the insight we give in this section should inform users and developers *a-priori* about how different component formulations, system compositions and media selections affect the existence of algebraic loops, and how such algebraic loops can be avoided. Based on these discussions, we also provide recommendations for efficient component model formulation.

[3] Specifically, FMU-CS does not allow calling `fmi2SetReal` followed by `fmi2GetReal` without first invoking `fmi2DoStep` (see p. 104 of the standard). Furthermore, `fmi2DoStep` does not allow a communication step size of 0.

Questions that this sections answers are:
1. Suppose we know the mass flow rate at each flow leg. Under which arrangements do FMUs, each representing a steady-state fluid flow component, cause an algebraic loop?
2. How does the answer to the above question change if computing the value of the mass flow rate requires solving a flow rate versus pressure drop calculation?
3. Under what conditions does the use of the regularization function to treat near zero mass flow rates cause algebraic loops, and how can they be avoided?

In the next section, we discuss direct input-output dependencies in major HVAC components, and afterwards discuss situations where these components are connected to form HVAC systems.

4.1 Major HVAC components

This section describes the direct input-output dependencies of major HVAC components under the assumption that they are modeled steady-state, as is common in building energy simulation. The purpose of the discussion in this section is to understand what outputs depend directly on what inputs and how direct dependencies can be reduced.

4.1.1 Heater

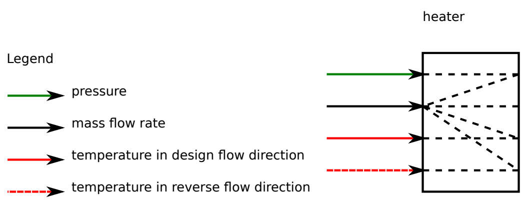
heater
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction

Figure 3. FMU of a heater.

We will start with a simple component of a heater that injects a known amount of heat $\dot{Q}$ into a fluid stream. In such a component, the outlet pressure is $p_b = p_a + f(\dot{m}_a)$ for some function $f : \mathbb{R} \to \mathbb{R}$, and the outlet temperature is $T_b = g(\dot{m}_a, \iota(T_a))$ for some function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. For example, for an ideal water heater, $g(\dot{m}_a, \iota(T_a)) = \iota(T_a) + \dot{Q}/(\dot{m}_a c_p)$. We will depict graphically such a component as shown in Figure 3, where the arrows indicate inputs (for this component, inputs are on the left and outputs on the right). The dotted lines inside the component show on what inputs an output directly depends on. We selected to use this graphical representation rather than writing the incidence matrix as the graphical representation allows us to graphically connect components to form HVAC systems.

4.1.2 Dehumidifying or humidifying components

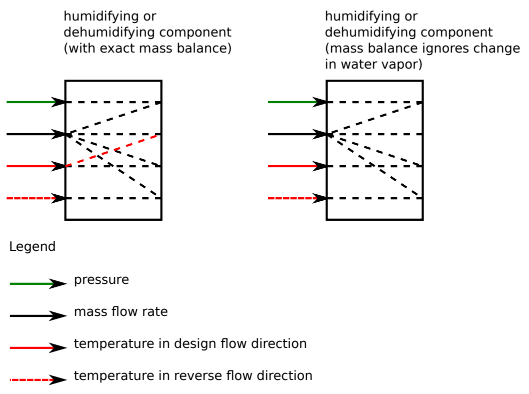
humidifying or dehumidifying component (with exact mass balance)
humidifying or dehumidifying component (mass balance ignores change in water vapor)
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction

Figure 4. FMU of a humidifying or dehumidifying component. The component on the right implements Recommendation 4.2, and hence the red dashed line is removed.

We now discuss the situation in which the heat exchanger in Figure 3 dehumidifies or humidifies the air. This can be the case for a humidifier or for a cooling coil that cools the air below its dew point. In this situation, the rate of heat and mass transfer affect the outlet mass flow rate. Therefore, if the outlet mass fraction of the humidifying or dehumidifying component depends on the thermodynamic state of the inlet fluid, which generally is the case, then the pressure drop equations are coupled to the heat transfer equations. Hence, such a component has the structure shown in Figure 4. Note that for this component, there generally is no need to properly characterize its thermodynamic behavior for reverse flow because such equipment is only operated when the fan is on. When the fan is off, small reverse flows may occur, but for this situation, it suffices to set $T_a = \iota(T_b)$ and $X_a = \iota(X_b)$, or $T_a = T_{default}$ and $X_a = \iota(X_{default})$, where $T_{default}$ and $X_{default}$ are default values for temperature and mass fraction. The latter version can lead to smaller system of equations if components are used in a flow network.[4] We therefore decided in Figure 4 that the change in mass flow rate has the functional form $\dot m_b - \dot m_a = f(\dot m_a, \iota(T_a), \iota(X_a))$ rather than $\dot m_b - \dot m_a = f(\dot m_a, \iota(T_a), \iota(X_a), \iota(T_b), \iota(X_b))$. I.e., the thermodynamic properties of the reverse flow are not used to compute the amount of humidification or dehumidification. We therefore make the following recommendation:

Recommendation 4.1 *To reduce the number of direct input-output dependencies of components that humidify or dehumidify the air, such components should implement for the reverse flow* `port_a.h_outflow=Medium.h_default`, *where* `Medium.h_default` *is the default specific enthalpy of the medium. Otherwise, the energy equations for the backward flow become part of the residual functions of the pressure and mass flow rate equations.*

Because the outlet mass flow rate is $\dot m_b = \dot m_a(1 + \Delta X_w)$, where ΔX_w is the change in water vapor mass fraction across the component, this component couples the energy calculation to the pressure drop versus mass flow rate calculations. However, in typical building HVAC systems, $\Delta X_w < 0.005\,\mathrm{kg/kg}$. Hence, by tolerating a relative error of 0.005 in the mass balance, one can decouple these equations. Decoupling these equations avoids having to compute the energy balance of the humidifier and its upstream components when solving for the pressure drop of downstream components[5]. We therefore make the following recommendation:

Recommendation 4.2 *If an error in the mass balance of about 0.5% is acceptable, then one can implement a humidifier or dehumidifier that neglects in the mass balance equation the change in water vapor mass fraction. This can allow computing the mass flow rate versus pressure drop equations without having to couple the energy balance, or the control input of a humidifier or dehumidifier, to these equations.*

As in building simulation, there is considerable uncertainty in air flow rate calculations, and also because larger effects such as duct leakage are generally ignored, taking a relative error of 0.5% into account seems acceptable in typical applications. See also Jorissen et al. (2015) for a discussion.

4.1.3 Fan

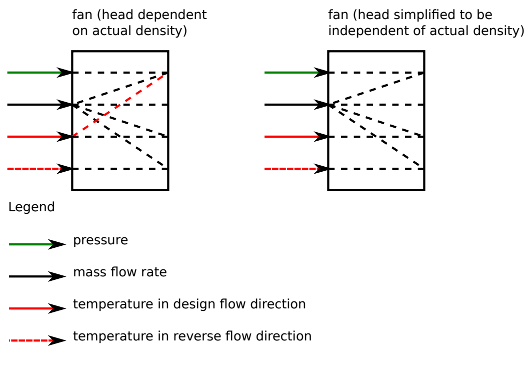
fan (head dependent on actual density)
fan (head simplified to be independent of actual density)
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction

Figure 5. FMU of a fan. The component on the right implements Recommendation 4.3, and hence the red dashed line is removed.

According to the laws of fluid dynamics, the pressure rise over a fan is related to the volume flow rate rather than the mass flow rate. Therefore, the functional form for the fan head is $p_b - p_a = f(\dot m_a, \iota(T_a), \iota(X_a))$ and the input-output dependency is as shown in the left-hand side of Figure 5. However, if one were to simplify

[4]See `https://github.com/iea-annex60/modelica-annex60/issues/281` for a discussion.

[5]In the `Buildings` library, only downstream components are affected because the humidifier evaluates a component's pressure drop for $\dot m_a$ and not for $\dot m_b$.

the fan laws and use a constant mass density, then the direct input-output dependency of the inlet thermodynamic properties could be eliminated, as shown in the right-hand side of Figure 5. We therefore make the following recommendation:

Recommendation 4.3 *If the operating temperature of a fan (or pump) does not change much, or if large uncertainties exist in parameters or the models for the pressure drop calculation of the duct (or pipe) network, then one should assume a constant mass density in the fan model, as this leads to fewer coupled equations.*

4.1.4 Heat exchanger between supply and return air

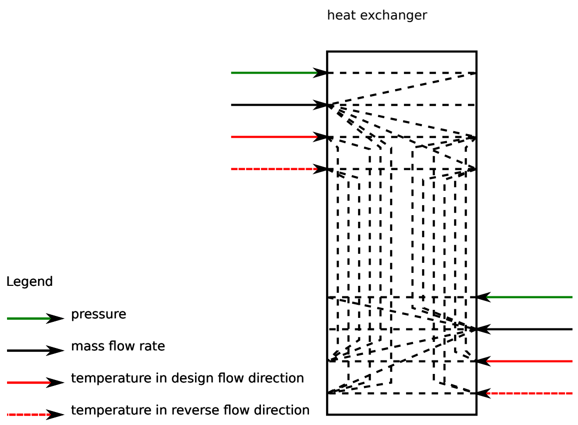
heat exchanger
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction

Figure 6. FMU of a component that exchanges heat between two fluid streams.

Figure 6 shows the direct input-output dependency of a heat exchanger. On top left is the inlet of one fluid stream and on the bottom right is the inlet of the other fluid stream. Such heat exchangers are typically modeled using two different implementations.

1. The simplest form is a constant effectiveness heat exchanger. In this situation, the rate of heat transfer is $\dot{Q} = \varepsilon \dot{C}_{min} (T_{in,1} - T_{in,2})$, where $\varepsilon \in (0,1)$ is a constant, $\dot{C}_{min} = \min(|\dot{m}_{a,1} c_{p,1}|, |\dot{m}_{a,2} c_{p,2}|)$ is the minimum heat capacity flow rate and $T_{in,1}$ is the inlet temperature of the fluid 1, which is equal to $\iota(T_{a,1})$ or $\iota(T_{b,1})$. Hence, the in-streaming thermodynamic properties of the forward and reverse flow must be known in order to compute the thermodynamic properties of the out-streaming fluid for forward and reverse flow.
2. A more elaborate model is one that uses the $\varepsilon - \text{NTU}$ model. In this situation the same direct input-output dependency is obtained as for the model with constant effectiveness.

This discussion shows that heat exchangers lead to complex direct input-output dependencies. If one were to compromise on not being able to properly compute the heat transfer if one or both streams reverse their direction, then one could simplify the model to the form shown in Figure 7. Here, we changed the model so that the transferred heat is zero if any of the flows is different from the design flow direction. Initial experiments indeed confirmed that such a simplified implementation leads to smaller systems of coupled equations. We there-

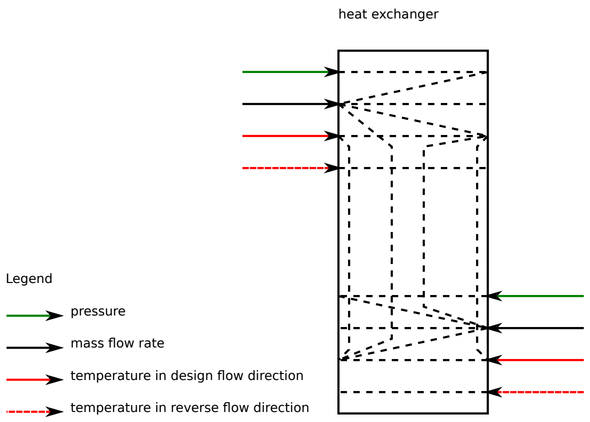
heat exchanger
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction

Figure 7. FMU of a component that exchanges heat between two fluid streams but with the simplification that no heat is exchanged if any of the flows is different from the design flow direction.

fore make the following recommendation:

Recommendation 4.4 *If the thermodynamic behavior of a heat exchanger under reverse flow directions is not of interest to the application, then the equations should only be formulated for forward flow. For reverse flow situations, one should simply assign $T_{b,k} = \iota(T_{a,k})$ for both streams $k \in \{1,2\}$. Note that reverse flow may occur in HVAC systems due physical reasons such as wind pressure on the facade (when the fan is off) or due to numerical artifacts because numerical solvers only compute an approximate numerical solution and hence small negative flows can exist when the HVAC system is off.*

4.1.5 Temperature or humidity control

The user guide `Annex60.Fluid.Sensors.UsersGuide` and of libraries that use `Annex60`, such as `AixLib`, `Buildings`, `BuildingSystems` and `IDEAS`, recommend to measure temperature, relative humidity, mass fraction, trace substances and specific enthalpy with a sensor that has two ports, and use a dynamic balance to compute the measured quantity. This dynamic balance has shown to be beneficial in large systems that can have zero flow rate. If such sensors are used as an FMU, they have the advantage that the dynamic balance causes the measured quantity to be a state variable. Hence, if used in combination with a P or PI controller, the use of this state variable avoids having to solve an algebraic loop.

Therefore, in the subsequent discussion, we will assume that a dynamic sensor is used.

4.2 Components in series

Figure 8 shows four FMUs in series. This represents the case where a mass flow rate of outside air is conditioned and transported to a room that has a dynamic

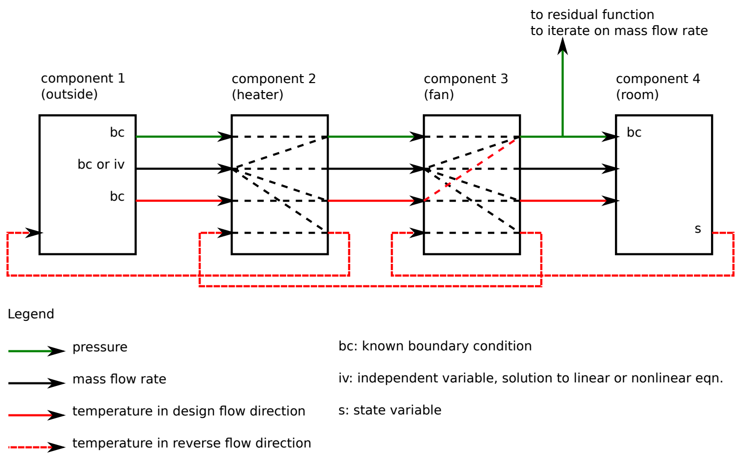
to residual function
to iterate on mass flow rate
component 1
(outside)
component 2
(heater)
component 3
(fan)
component 4
(room)
bc
bc or iv
bc
bc
s
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction
bc: known boundary condition
iv: independent variable, solution to linear or nonlinear eqn.
s: state variable

Figure 8. FMUs connected in series.

energy balance, and hence the room air temperature is a state variable whose value is determined by an integration algorithm. The outside air imposes a pressure and temperature boundary condition. The outdoor mass flow rate is either an independent variable or a boundary condition. In the first case, the outdoor mass flow rate is iterated upon until the pressure equations are satisfied. In the second case, the pressure drop equations could be removed from the set of equations.

The equations for this arrangement can be solved as follows. First, by assumption, the pressure drop only depends on the mass flow rate and not on temperature. Therefore, the mass flow rate can be solved iteratively by setting an initial value, evaluating the pressure drop equations of component 1, 2, 3 and 4 in series until a convergence criteria on the difference between the outlet pressure of component 3 and the room air pressure (component 4) is met. Once the mass flow rate is known, components 1, 2, 3 and 4 can be called in sequence to obtain the temperatures for the forward flow direction. For the reverse flow direction, components 4, 3, 2 and 1 need to be called in sequence.

The red line in the fan of Figure 8 is only present if Recommendation 4.3 is not implemented. In this situation, the energy equation of the heater need to be evaluated in order to compute the mass flow rate, thereby increasing the number of operations required to evaluate the residual function.

Analyzing Figure 8 leads us to the following remark:

Remark 4.1 *Evaluating the energy equations for forward flow and then for backward flow is only possible if the energy equations only depend on the thermodynamic state of the inflowing medium. For example, if a component were to use the regularization*

```
h_in = spliceFunction(
          pos    = inStream(port_a.h_outflow),
          neg    = inStream(port_b.h_outflow),
          x      = m_flow,
          deltax = m_flow_nominal/100)
```

then the thermodynamic properties of the backward flow must be known to compute the thermodynamic properties of the forward flow. Moreover, if, in Figure 8, components 2 and 3 both use the above `spliceFunction`*, then a nonlinear equation must be solved to compute the thermodynamic properties.*

Hence, we make the following recommendation:

Recommendation 4.5 *Regularization in which the arguments of the regularization function directly depend on the thermodynamic properties of the forward and reverse flow should be avoided as this can lead to nonlinear equations.*

Note, however, the following:

Remark 4.2 *Simply replacing*

```
h_in = spliceFunction(
          pos    = inStream(port_a.h_outflow),
          neg    = inStream(port_b.h_outflow),
          x      = m_flow,
          deltax = m_flow_nominal/100)
```

with

```
h_in = if m_flow >= 0
          then inStream(port_a.h_outflow),
          else inStream(port_b.h_outflow);
```

is not a solution to Recommendation 4.5. In fact, this would also lead to a non-linear equation, but with a discontinuity in the residual equation, which can lead to problems in Netwon-Raphson solvers. Rather, one could attempt to set `h_in=inStream(port_a.h_outflow)` *and let the transfered heat go to zero as the mass flow rate approaches zero from above.*

4.3 Components in parallel

Figure 9 is as Figure 8 except that it has two rooms, each with a variable air volume (VAV) terminal. The VAV terminals can increase the flow resistance based on a control signal, and possibly provide heating or cooling (here, we assumed no dehumidification at the terminal unit). To implement such a system, a flow splitter is needed. The flow splitter has as an input the split of the mass flow fraction between the two outlet ports. This input is required as otherwise the splitter is underdetermined.

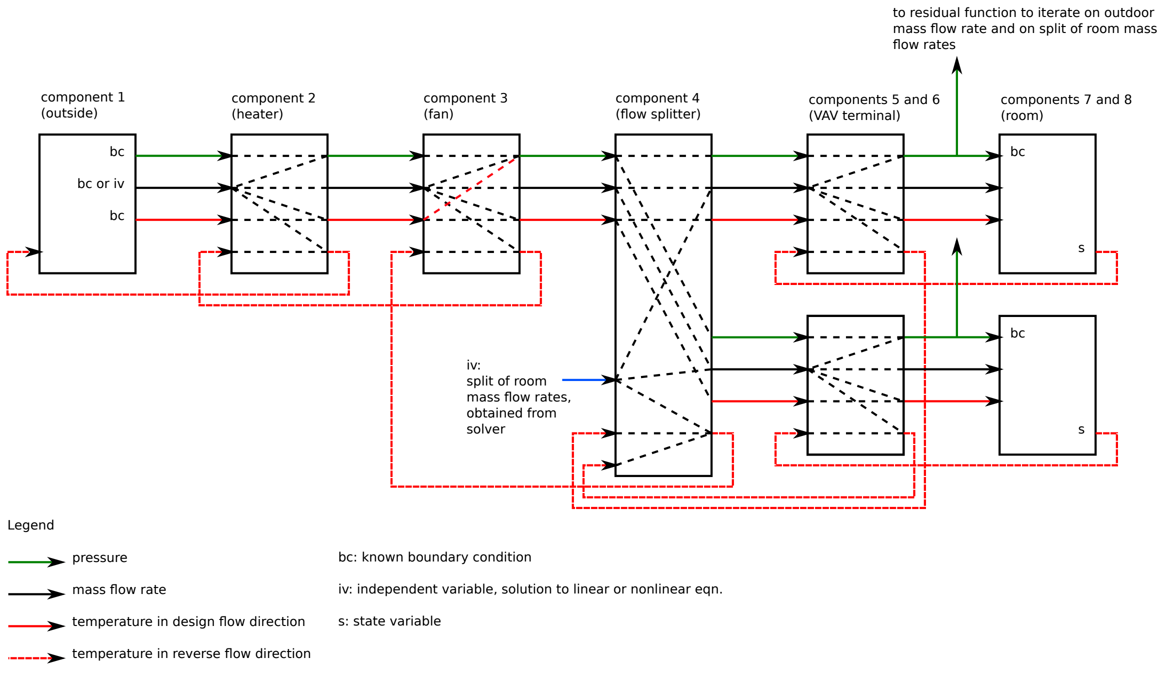
to residual function to iterate on outdoor mass flow rate and on split of room mass flow rates
component 1 (outside)
component 2 (heater)
component 3 (fan)
component 4 (flow splitter)
components 5 and 6 (VAV terminal)
components 7 and 8 (room)
bc
bc or iv
bc
s
iv: split of room mass flow rates, obtained from solver
Legend
pressure
mass flow rate
temperature in design flow direction
temperature in reverse flow direction
bc: known boundary condition
iv: independent variable, solution to linear or nonlinear eqn.
s: state variable

Figure 9. FMUs connected in series that serve two rooms.

The input may be a variable determined by a numerical solver. The VAV terminal has the same input-output relation as the heater.

The equations for this arrangement can be solved as follows: A solver determines the outdoor mass flow rate and the split of the mass flow rates until the residual function of the pressure is satisfied. This can be accomplished by evaluating the pressure drop equations of all FMUs along the flow direction. If the fan does not implement Recommendation 4.3, then the energy equation of components 2 and 3 also need to be evaluated. Once the mass flow rate has converged to its solution, components 1, 2, 3, 4, 5, 6, 7 and 8 can be evaluated for forward flow. Finally, components 7, 8, 5, 6, 4, 3, 2 and 1 can be evaluated for reverse flow.

4.4 Air loops

Figure 10 shows an air loop that consists of the heat exchanger that implements Recommendation 4.4, two fans that implement Recommendation 4.3, and a return duct. The room conserves mass and has a pressure equation for the outlet pressure. Therefore, during the iterative solution of the mass flow rate, convergence is checked on the pressure of the exhaust air.

The equations can be solved as follows: First, components 1, 2, 3, 4, 5, 6 and again 2 are evaluated to solve for the mass flow rate. Next, the energy equation can be solved for forward flow by evaluating components 5 (to get the state T) and 6 to obtain the return air inlet temperature of the heat exchanger. Then, components 1, 2, 3, 4 and 5 can be evaluated, which concludes the computations for the forward flow. For the reverse flow direction, components 1, 5, 4, 3, 2, 6 and again 1 and 5 can be evaluated. Note that the order is not unique as one could have started with component 5.

Remark 4.3 *Note that the heat exchanger is called at least four times if flow reversal is allowed, i.e., twice for the iteration for the mass flow rate, once for forward flow and once for reverse flow. Without flow reversal, the heat exchanger is called at least three times. This indicates the inherent inefficiencies when using FMUs for individual fluid flow components, rather than letting the symbolic processor of a Modelica tool rearrange the equations to a block lower triangular form.*

4.5 Control Loops

As discussed in Section 4.1.5, feedback control loops for thermodynamic properties such as temperature or humidity do not cause an algebraic loop if a dynamic sensor is used. Specifically, if a sensor from the package `Buildings.Fluid.Sensor` is used and its time constant `tau` is set to a value larger than zero, then the sensor will output a state variable and hence the feedback control loop does not cause an algebraic loop. If `tau=0` and the controller has direct feedthrough, then such control loops for steady-state HVAC components cause an algebraic loop. To avoid such algebraic loops, the controller could be idealized and implemented directly in the HVAC component, as is done for example in the model `Buildings.Fluid.HeatExchangers.HeaterCooler_T`.

5 Connector variables

In this section, we discuss the selection of the variables that will appear as inputs and outputs of the FMU. We recall that `Modelica.Fluid`, `Annex60.Fluid` and libraries that depend on it such as `AixLib`,

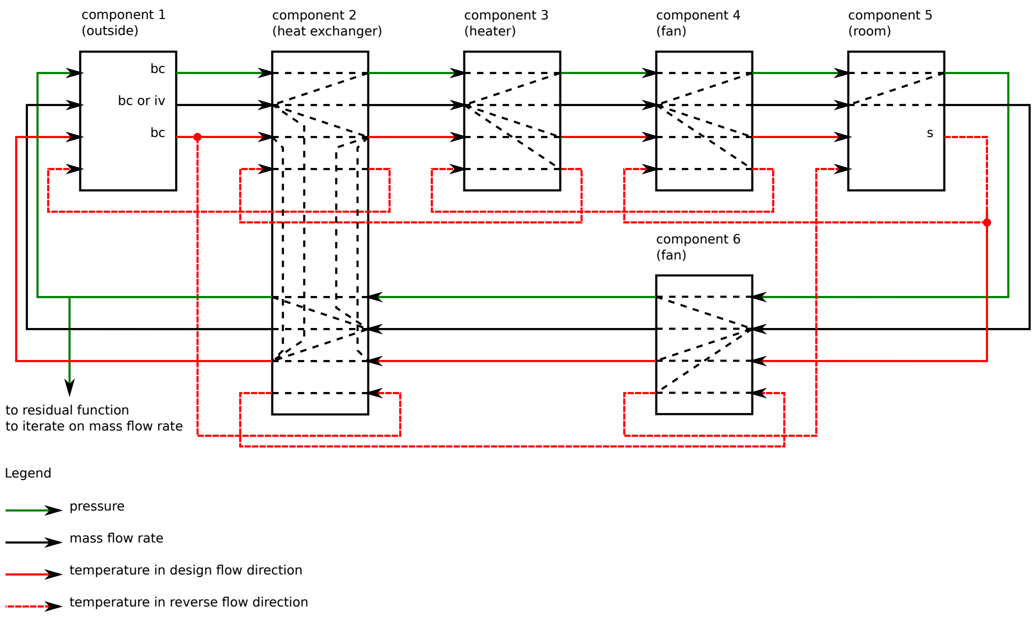

Figure 10. FMUs connected in a loop with a heat exchanger between the outside air intake and the exhaust air.

`Buildings`, `BuildingSystems` and `IDEAS`, use for the pressure the total pressure in Pascal, for the mass flow rate kg/s and for the mass concentration for moist air kg/kg total air. We will use the same variables for the parameters, input and output signals of the FMU. While the connectors in the above libraries use specific enthalpy h, we will use the absolute temperature T in Kelvin instead. The reasons for this selection are as follows:

1. If we were to use the specific enthalpy h as a connector variable, then an FMU would not be self-contained. Rather, to use the FMU, one would require knowledge of the function that is used by the FMU to relate temperature and mass fraction to enthalpy. Consequently, exchanging models in FMUs would not be possible without also providing such a function.
2. In Modelica, using h is motivated as it allows mixing of fluid streams in a port, e.g., in a port, $h_{mix} = \sum_i \max(0, \dot{m}_i)\, h_i / \sum_i \max(0, \dot{m}_i)$, where h_i is the enthalpy of the fluid that flows into the port. Using temperature in the mixing equations that are generated by the fluid connections can give wrong results as $T_{mix} = \sum_i \max(0, \dot{m}_i)\, T_i / \sum_i \max(0, \dot{m}_i)$ only holds if the specific heat capacity c_p is constant. However, this is not a concern for FMUs as mixing in ports is supported in Modelica but not when FMUs are connected among each other.

We also had to make a choice about using Kelvin or degree Celsius for the temperature. Whereas users may be more accustomed to use degree Celsius, we decided to use Kelvin for the following reasons:

1. FMUs for model-exchange and for co-simulation not only expose input and output signal, but also state variable and parameters. The state variables in models of the `Buildings` library are temperature in Kelvin. Changing them to degree Celsius would require redesigning the library, and hence using a unit convention in the `Buildings` library that is different from what is used in the Modelica Standard Library.
2. Without such a redesign, FMUs would require some temperatures in Celsius and others in Kelvin.
3. Many models have parameters for design temperatures, and also compute outputs that are temperatures, such as temperature sensors or the temperature of a furnace. These quantities have units of Kelvin. Hence, for all parameters and all such signals, a unit conversion would need to be implemented, which would be quite cumbersome. Moreover, such parameters and variables may still show up as an FMU interface variable, thereby introducing mixed units.

Because using mixed units is confusing and error-prone, we use Kelvin and propose that tools handle unit conversions between the computed quantities and the quantities that are displayed to the user, as is done for example in Dymola 2016.

With these design decisions, an FMU that has two fluid ports called `inlet` and `outlet` will have the following interface variables.

```
inlet.m_flow
      p
      forward.T
              X_w
              C
      backward.T
               X_w
               C
```

where `m_flow` is the mass flow rate, `p` is the absolute pressure (which is conditionally removed if `use_p_in=false`) and `forward` and `backward` are the thermodynamic properties for the forward flow and backward flow. If `allowFlowReversal=false`, then `backward` is removed. The thermodynamic variables are temperature `T` in Kelvin, water vapor mass fraction `X_w` in kg/kg total air, which is removed for water, and trace substances `C`, which is removed if `Medium.nC=0`.

6 Code optimizations lost by using small FMUs

This section describes code optimizations that are no longer done when models are shared as an FMU as opposed to sharing Modelica mode, because FMUs either contain compiled code or C-code, neither of which allows the level of computer algebra possible with Modelica. While the Modelica specification does not prescribe the code optimization, most Modelica compilers are expected to conduct the optimizations described below.

1. Consider Figure 8. A Modelica compiler would use one variable for the mass flow rate that enters the component, and one for the leaving mass flow rate. Results can be written efficiently by storing only the mass flow rate $\dot{m}_{1,b}$ that leaves component 1, and declaring in the output file that $\dot{m}_{k,b} = \dot{m}_{1,b}$ for $k \in \{2,3\}$ and $\dot{m}_{k,a} = \dot{m}_{1,b}$ for $k \in \{2,3,4\}$. However, such knowledge is no longer available if multiple FMUs are used. Hence, mass flow rates must be set, read, stored and written to disk multiple times. Similar discussions apply for thermodynamic properties that remain unchanged in a component, such as T, X_w and C in an air damper.
   The efficiency loss that incurs if the output has the same value as the input could, however, be avoided using optional features of the FMI standard. For example, first, if variables share the same `valueReference` in the `modelDescription.xml` file, then they have the same value. Second, if `dependenciesKind="fixed"` is declared in the `modelDescription.xml` file, then the output is, after `fmi2ExitInitializationMode`, equal to a fixed factor times the input, and hence a master algorithm can deduce that they are equal. Dymola 2016 uses the latter construct.
2. Consider Figure 8. If $\dot{m}_{1,b}$ is an iteration variable, components 2 and 3 can be configured to compute pressure drop as a function of the mass flow rate, rather than mass flow rate as a function of the pressure drop, thereby keeping the number of iteration variables as small as possible. Such a selection is no longer possible if a component is exported as an FMU. See also Jorissen et al. (2015) for how this can affect computing time.
3. Consider Figure 8. A Modelica compiler may do automatic differentiation of the Modelica code to compute a symbolic expression of the Jacobian matrix that is used to iteratively solve for the mass flow rate that satisfies the constraint on the pressure.
4. In Figure 9, if the VAV terminals 5 and 6 both require the evaluation of psychometric functions that depend on its inlet temperature and humidity, which are equal for components 5 and 6, then Modelica compilers can compute these functions once and assign the results to both components using what is called common subexpression evaluation.
5. If no pressure drop calculation is requested, in Modelica it is possible to remove all these computations. In FMUs, while computations can be disabled with a boolean parameter, there will still be an input-output dependency, causing a system simulator to wrongfully believe that there is an algebraic loop.
6. If multiple components form a system of equations, Modelica compilers may solve it explicitly if it is small and linear. If it is nonlinear, a Modelica compiler can use block-lower triangularization and tearing to reduce its dimension (Cellier and Kofman, 2006).

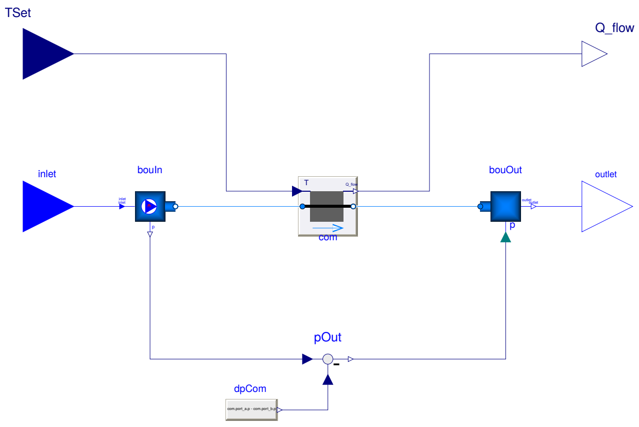

Figure 11. Block that contains a replaceable model of a heater and that defines input and output signals for export as an FMU.

As a drawback when allowing these optimizations, one would require a Modelica translator and a C-compiler on the host simulator. Also, for large Modelica models that involve buildings and HVAC systems, translation and compilation time can be in the order of minutes in tools such as Dymola or OpenModelica. This, however, could be reduced by compiling only the HVAC system, and by doing incremental parallel compilation.

7 Implementation

We implemented the package `Buildings.Fluid.FMI` that contains connectors, blocks that have replaceable thermofluid components, examples blocks that can be exported as FMUs, and examples in which we connected these example blocks to form system models.

Figure 11 shows such an example block. In the middle is the actual thermofluid component. In this case, it is a heater or cooler, but it may be a whole subsystem that contains multiple thermofluid components as long as it extends `Buildings.Fluid.Interfaces.PartialTwoPort`. To the left and right are adaptors that convert between the signal flow and the acausal fluid connectors. At the bottom is the computation of the pressure difference across the component. This is required as one adaptor needs to set the flow rate and the other the pressure in order for the component to be balanced. The connectors `inlet` and `outlet` contain the input and output signals. The `inlet` connector is defined as follows (most annotations have been removed for better readability):

```
within Buildings.Fluid.FMI.Interfaces;
connector Inlet "Connector for fluid inlet"
  import FMI = Buildings.Fluid.FMI;
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium
    "Medium model";
  parameter Boolean use_p_in = true
    "= true to use a pressure from connector";
  parameter Boolean allowFlowReversal = true
    "= true to allow flow reversal";
  input Medium.MassFlowRate m_flow
    "Mass flow rate into the component";
  FMI.Interfaces.PressureInput p
    if use_p_in
    "Thermodynamic pressure";
  input FMI.Interfaces.FluidProperties
    forward(
      redeclare final package Medium = Medium)
      "Inflowing properties";
  output FMI.Interfaces.FluidProperties
    backward(
      redeclare final package Medium = Medium)
      if allowFlowReversal
      "Outflowing properties";
end Inlet;
```

The connector `Buildings.Fluid.FMI.Interfaces.FluidProperties` contains the thermodynamic properties, and is defined as follows:

```
within Buildings.Fluid.FMI.Interfaces;
connector FluidProperties
  "Type definition for fluid properties"
  import FMI = Buildings.Fluid.FMI;
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium
      "Medium model";
  Medium.SpecificEnthalpy h
    "Specific thermodynamic enthalpy";
  FMI.Interfaces.MassFractionConnector X_w
    if Medium.nXi > 0
    "Water vapor mass fractions per kg total air";
  Medium.ExtraProperty C[Medium.nC]
    "Properties c_i/m";
end FluidProperties;
```

Note that we introduced the new connectors `Buildings.Fluid.FMI.Interfaces.PressureInput` and `Buildings.Fluid.FMI.Interfaces.MassFractionConnector`. The first was required to conditionally remove the pressure from the connector. For example, if a user is not interested in computing the pressure drop, then setting the parameter `use_p_in=false` will eliminate `p` from the connector, remove all pressure drop calculations and setting the pressure of the component to `Medium.p_default`. We also decided to introduce the new connector

```
within Buildings.Fluid.FMI.Interfaces;
connector MassFractionConnector =
  Modelica.SIunits.MassFraction
  "Water vapor mass fraction per kg total mass";
```

to avoid having a vector with one component for the water vapor mass fraction. This was done so that the FMUs have as an input or output for the water mass fraction a scalar variable `X_w` rather than having a vector with one component for the water vapor mass fraction.

In the `Buildings` library, when running the regression tests, for each model that is exported as an FMU a file will be generated that shows the dependencies of outputs, states and initial unknowns. This file can be inspected to see what dependencies thermofluid flow components have, and the file will be used in subsequent regression tests to verify that the dependencies do not inadvertently change when models are revised.

8 Numerical experiments

8.1 Connector Variables

To benchmark the computing time with temperature T versus specific enthalpy h in the FMU input and output, we simulated an HVAC system. The HVAC system is a VAV system with economizer, heating and cooling coil in the air handler unit, and models of return duct, splitter, terminal heaters and controls. The FMUs either had T or h as input and output variables. Internally, the models use enthalpy balance, and hence if T is an input and output variable, a conversion from T to h is required for the input and from h to T for the output. We exported the components as nine FMUs from Dymola 2015 FD01 and connected and simulated them in Ptolemy II (Ptolemaeus, 2014) for the same number of steps. To bridge from Java used in Ptolemy II to FMI, we used the Java Native Interface (JNI). This experiment did not show a difference in computing time.

Next, we simulated the Modelica implementations with T or h in the inlet and outlet signals, connected to a first order room response, directly in Dymola with the Rkfix3 integration algorithm, without use of any FMUs. This experiment also showed no difference in computing time.

These two experiments indicate that there is no performance penalty of choosing T rather than h for the input and output signals.

8.2 Code optimizations lost by using small FMUs

To investigate the impact of lost code optimization, we simulated the HVAC model of Section 8.1 that was exported as either one or as nine FMUs. Both systems were simulated in Ptolemy II for 35,040 steps, which would correspond to an annual simulation with an average time step of 15 minutes. The simulation time was 2 seconds for the case with one FMU, and 3.4 seconds for the case with nine FMUs. Hence, using nine FMUs increased the computing time by 70%. The difference is attributed to the lost code optimization in FMUs, the overhead of calling many FMUs, and transferring data between outputs and inputs of FMUs. Note however that the version of Ptolemy II that we used for our experiments does not make use of the `dependenciesKind` information explained in Section 6, item 1.

9 Conclusions

The analysis in Section 4 showed that regularization and the use of the `inStream` function can cause direct input-output dependencies in FMUs that contain steady-state HVAC equipment models. Recommendations to avoid such dependencies are provided. We also provided various recommendations to implement approximate equations in thermofluid flow models that lead to fewer input-output dependencies, and hence smaller coupled systems of equations.

Our analysis showed that using multiple small FMUs prevents system-level code optimization that is otherwise done in Modelica. This was confirmed by our numerical experiments.

For users, we provide a Modelica package that allows export of thermofluid flow components and systems for different media, with and without pressure drop calculations.

In summary, the efficiency of using FMUs for thermofluid flow components strongly depends on component design, and various code optimizations are lost when using small FMUs rather than Modelica models.

10 Acknowledgment

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and Building Information Modeling standards.

References

François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.

Rüdiger Franke, Francesco Casella, Martin Otter, Michael Sielemann, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Stream connectors – an extension of modelica for device-oriented modeling of convective transport phenomena. In Francesco Casella, editor, *Proc. of the 7-th International Modelica Conference*, Como, Italy, September 2009. Modelica Association. URL `https://www.modelica.org/events/modelica2009/Proceedings/memorystick/pages/papers/0078/0078.pdf`.

Filip Jorissen, Michael Wetter, and Lieve Helsen. Simulation speed analysis and improvements of Modelica models for building energy simulation. In *11-th International Modelica Conference*, Paris, France, September 2015. Modelica Association.

Modelica, 2010. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.2*. Modelica Association, March 2010. URL `https://www.modelica.org/documents/ModelicaSpec32.pdf`.

Modelica Association. *Functional Mock-up Interface for Model-Exchange and Co-Simulation version 2.0*, 2014. `https://www.fmi-standard.org/downloads`.

Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. URL `http://ptolemy.org/books/Systems`.

Michael Wetter, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:10.1080/19401493.2013.765506.

Michael Wetter, Thierry S. Nouidui, David Lorenzetti, Edward A. Lee, and Amir Roth. Prototyping the next generation energyplus simulation engine. Accepted: *13-th IBPSA Conference*. International Building Performance Simulation Association, December 2015. URL `http://www.ibpsa.org/`.