Simplifications for Hydronic System Models in Modelica

F. Jorissen[a,c] M. Wetter[b], L. Helsen[a,c]
[a]University of Leuven, Department of Mechanical Engineering,
Celestijnenlaan 300 - bus 2421, 3000 Leuven, Belgium;
[b]Lawrence Berkeley National Laboratory, Energy and Environmental Technologies Division,
1 Cyclotron Rd 90-3147, Berkeley, CA, USA;
[c]EnergyVille, Thor Park 8310,
3600 Genk, Belgium;

Abstract

Building systems and their heating, ventilation and air conditioning flow networks, are becoming increasingly complex. Some building energy simulation tools simulate these flow networks using pressure drop equations. These flow network models typically generate coupled algebraic nonlinear systems of equations, which become increasingly more difficult to solve as their sizes increase. This leads to longer computation times and can cause the solver to fail. These problems also arise when using the equation-based modelling language Modelica and Annex 60 based libraries. This may limit the applicability of the library to relatively small problems unless problems are restructured. This paper discusses two algebraic loop types and presents an approach that decouples algebraic loops into smaller parts, or removes them completely. The approach is applied to a case study model where an algebraic loop of 86 iteration variables is decoupled into smaller parts with a maximum of 5 iteration variables.

1 Introduction

Building Heating, Ventilation and Air Conditioning (HVAC) systems are becoming increasingly complex due to new requirements such as the use of new energy-efficient heating and cooling systems with increasing share of renewable energy sources and more stringent ventilation standards. For detailed building energy simulations these flow networks can be modelled using first-principle physical models. The fluid flow direction and magnitude in a system are governed by the pressure drops that are generated by viscous friction in various components and by the pressure heads that are generated by pumps. Building Energy Simulation (BES) tools often neglect pressure drops and only consider the flow rates. HVAC systems are however becoming increasingly complex, with multiple actuators such as valves and variable-speed pumps that aim to improve the controllability of the system, while reducing electrical power use. To simulate these types of systems more accurately, simulation models can be extended such that they compute pressure drop based on flow rates. The added accuracy however incurs a higher computational cost. Consider for instance a hydronic system containing n components that model nonlinear flow friction. For such systems in the order of n pressure drops are computed. Since flow networks are typically considered to be quasi-stationary, a non-linear algebraic loop with up to n iteration variables may be formed if the problem structure

is not exploited. These algebraic loops are typically solved using Newton-type solvers, which use a Jacobian computation containing up to n^2 elements. As the inverse of the Jacobian is needed, the computation time can scale as $O(n^3)$. This results in a computation time that scales badly with the problem size (Jorissen, Wetter, and Helsen, 2015). Moreover, as convergence of Newton solvers depends on the initial guess, Newton solvers may not converge.

Modelica is an equation based modelling language that allows simulation of multi-physical systems (Mattsson, Elmqvist, and Broenink, 1997) and is gaining interest in the field of building simulation. A number of open source Modelica libraries such as IDEAS (Baetens et al., 2015), Buildings (Wetter et al., 2014), AixLib (Müller et al., 2016) and BuildingSystems (Nytsch-Geusen et al., 2013) allow the simulation of buildings, including pressure-driven water and air flow circuits. At the core of these libraries is the Annex 60 library (Wetter et al., 2015) (Wetter and van Treeck, 2017), which provides models for fluid flow networks. Annex 60 development is continued as part of IBPSA project 1.

The goal of this paper is to provide a comprehensive overview that explains when two main types of algebraic loops in flow networks are formed if pressure drop is computed as a nonlinear function of the flow rate and how they can be avoided. This discussion should allow the Modelica *user* to gain more insight into the Modelica toolset, allowing them to simulate larger flow networks by manipulating the problem structure. The Modelica library *developer* can learn how to structure models such that fewer or smaller algebraic loops are generated. The discussion is based on the Modelica tool chain, but the insights may also be applicable to other simulation tools and problems that are governed by the same underlying mathematical problems. This work is complementary to an earlier publication (Jorissen, Wetter, and Helsen, 2015). More specifically, Jorissen, Wetter, and Helsen (2015)present broadly applicable simplifications, applied to simple models and they state that such simplifications can lead to 2 to 3 orders of magnitude speed increase for a large model. This paper further discusses in detail how these simplifications are applied to the hydronic network of the larger model. For the Modelica tool developer, possible points of improvement are identified that are specific to the BES field.

This paper is structured as follows: Section 2 first provides background information that is required to understand the rest of the discussion. Some background information regarding Modelica is provided, main Annex 60 library equations are presented and algebraic loops are discussed. Section 3 outlines the methodology, which consists of three approaches that can be used to simplify or eliminate algebraic loops and rules of thumb for modelling the thermal dynamics of hydronic systems. Section 4 applies the methodology to a specific case study where an algebraic loop with 86 iteration variables is split and simplified such that multiple algebraic loops with no more than 5 iteration variables are obtained. Section 5 presents the conclusions.

2 Assumptions and Background

We assume that flow friction is computed using a nonlinear function that depends on the mass flow rate. Hence, we neglect temperature dependency of density and viscosity. Further, we assume the flow distribution to be steady-state.

Algebraic loops are coupled systems of algebraic equations. The type and configuration of the used HVAC component model equations determines what algebraic loops exist, but not how they are solved, which is tool-specific. The user chooses the configuration and type of HVAC components. Therefore, knowing the relevant equations in the used component models and understanding when

algebraic loops are formed is key to understanding the following discussion. This section therefore provides background information. Firstly, basic Modelica concepts are explained. Secondly, a qualitative overview of the relevant equations in typical Annex 60 models is presented. Thirdly, we discuss how algebraic loops are typically identified and how they are solved.

2.1 Modelica background

Modelica models declare a number of variables and contain a number of equations from which the variables are solved. For pressure drop elements these equations require boundary conditions from outside the considered model, and conversely the variables that are defined in a model may become boundary conditions in a different model. To enable this exchange of information, Modelica models use ports. Different types of ports exist, e.g. a `RealInput`, `HeatPort` or `FluidPort`. A `RealInput` is used to input a real variable into a model. A `HeatPort` is used to exchange heat. It consists of one `potential` variable, the temperature T, and one `flow` variable, the heat flow rate $\dot{Q}$. A `FluidPort` represents a fluid connection and consists of variables representing the pressure at the port, which is a `potential` variable, the mass flow rate entering the model, a `flow` variable, and a `stream` variable representing the specific enthalpy of the fluid. A port may declare optional `stream` variables such as glycol concentrations and water vapour fractions. For each `stream` variable, two associated numerical values exist: one value assuming the fluid exits the model, which we denote by `s` and one value assuming the fluid enters the model, which is denoted using `inStream(s)`. See Franke et al. (2009) for more information regarding `stream` variables.

When connecting two or more ports to each other, a connection set is generated. For each connection set, equations are generated depending on the involved variable type (`potential`, `flow` or `stream`). The quantities corresponding to `flow` variables are conserved such that their values add up to zero. The values of `potential` variables are equated. For a connection set that consists of two ports 'a' and 'b', the value of the entering `stream` variable s of 'a' equals the exiting value of s of 'b' such that $\text{inStream}(s_a) = s_b$ and vice versa. When a connection set consists of n $>$ 2 ports then n mixing equations are generated such that

$$\text{inStream}(s_i) = f(s_j, \dot{m}_j) \qquad 8i \; 2 \; f1, \ldots, ng, \quad 8j \; 2 \; f1, \ldots, ngnfig, \tag{1}$$

where s_i and $\dot{m}_i$ are respectively the stream variable and the mass flow rate corresponding to port i, and $f(\ ,\ )$ is a function that conserves the product $\dot{m}_i\, s_i$ and that has a continuous derivative around zero mass flow rate $\dot{m}_i$ (Franke et al., 2009). When s is an enthalpy variable, then this equation corresponds to conservation of energy.

2.2 Annex 60 model equations

Table 1 contains an overview of some basic models in the Annex 60 library. The fluid ports of these models (blue or white-blue circles) may be used to connect models to each other. The equations that are introduced by connecting components from the *outside* were discussed in Section 2.1. We now discuss the relevant equations that exist *inside* of the models that are listed in Table 1. The purpose of this discussion is not to be complete, but rather to show the main variable dependencies that are relevant for this discussion. The total set of equations determines what algebraic loops are formed. This will be discussed in Section 2.3. We consider two groups of equations: equations associated to pressures and mass flow rates, and equations with properties that are carried by the flow, e.g., enthalpy and mass fractions.

In the following discussion, we assume that each component enforces conservation of mass such that the sum of the mass flow rates entering and exiting through all fluid ports equals zero. This

property is satisfied when using the default Annex 60 water `Medium`. For air media this may not be the case when air is configured to be compressible.

At this point, we note that in this paper, we are concerned with the situation in which the mass flow distribution is computed based on non-linear flow friction. This is how most users use Modelica. However, Annex 60-based libraries allow three options: Firstly, all flow resistance can be modelled using nonlinear functions of the flow rate, and for valves and dampers, the valve and damper position. This is the most detailed situation, which we examine in this paper. Secondly, the nonlinear flow resistances can be replaced by linear functions using a parameter, which greatly simplifies the flow network. Thirdly, the flow resistances could be removed by setting the pressure drop at design flow to zero. In this case, the flow rates can be prescribed in each flow branch, as it is typically done in building simulation tools such as EnergyPlus and TRNSYS. Moreover, we note that although EnergyPlus computes flow friction for hydronic plants, it does so by prescribing for each parallel flow branch the mass flow rate, and then adding what is called in its documentation an "imaginary valve". These imaginary valves adjust the pressure drop in order to get consistent pressure at the flow junctions. Hence, to avoid nonlinear systems of equations, EnergyPlus simplifies the physics by prescribing the flow rate and adding artificial pressure drop elements. Furthermore, it places various restrictions for where users are allowed to add pressure drop elements (U.S. Department of Energy, 2016).

2.2.1 `Pressure drop`

`PressureDrop` is a generic component for modelling pressure drops as

$$\Delta p = \text{sign}(\dot{m})(\dot{m}/k)^2 \tag{2}$$

where Δp is the pressure difference between the two fluid ports of the component, $\dot{m}$ is the mass flow rate through each of the fluid ports of the component and k is a flow coefficient that is computed using a nominal value for the pressure drop and the mass flow rate. Note that a regularisation around zero flow is implemented for $|\dot{m}| < \dot{m}_{turb}$, where $\dot{m}_{turb}$ is the mass flow rate around which turbulence occurs. See Wetter et al. (2015) for more details.

For computational reasons it sometimes is more efficient to use the inverse of (2) (Jorissen, Wetter, and Helsen, 2015). Therefore Annex 60 component models that generate pressure drop equations allow switching between (2) and its inverse,

$$\dot{m} = \text{sign}(\Delta p) k \sqrt{\text{abs}(\Delta p)}. \tag{3}$$

The switch is done using a boolean parameter `from_dp`. The `PressureDrop` model is isenthalpic. When the flow in a model never reverses, the exiting enthalpy value of the upstream port is never used, therefore its value can be set to anything, which may reduce model complexity. The user can flag this for most models using a boolean parameter `allowFlowReversal`. The implementation is

$$h_b = \text{inStream}(h_a), \tag{4}$$

$$h_a = \text{if allowFlowReversal then inStream}(h_b) \text{ else } h_0, \tag{5}$$

where h_b and h_a are the enthalpies leaving through the upstream and downstream port, and h_0 is a default enthalpy.

Table 1: Illustration and short description of the main component models that are used in the example models.

Icon	Description	Source
	`PressureDrop`: Pressure drop element that computes $\Delta p = \text{sign}(\dot{m}) \left(\frac{\dot{m}}{k}\right)^2$ or $\dot{m} = \text{sign}(\Delta p) k \sqrt{\text{abs}(\Delta p)}$ depending on the value of parameter `from_dp`.	Annex 60 v1.0 (Wetter et al., 2015)
	`MixingVolume`: This component mixes all incoming fluids and causes no pressure drop.	Annex 60 v1.0 (Wetter et al., 2015)
	`Two-way valve`: This computes the same equation as `PressureDrop`, but instead of a parameter, k is a function of the control signal of the valve.	Annex 60 v1.0 (Wetter et al., 2015)
	`Junction`: This component connects one instance of `PressureDrop` from each of the three ports to a `MixingVolume`, which can optionally be removed using a parameter. The three k values are computed from parameters.	Annex 60 v1.0 (Wetter et al., 2015)
	`Three-way valve`: This component internally connects one two-way valve between each of the black valve legs to a `MixingVolume`, which is also connected to the white valve leg. The `MixingVolume` can optionally be removed.	Annex 60 v1.0 (Wetter et al., 2015)
	`Boundary`: Used to set the absolute pressure at a point in the hydronic system.	Annex 60 v1.0 (Wetter et al., 2015)
	`Pump`: The pump used in this work sets Δp to a control input, or sets the speed N of the pump such that $\Delta p = f(\dot{m}, N)$, where N is the control input.	Annex 60 v1.0 (Wetter et al., 2015)
	Component that exchanges heat between two fluid streams and that uses one `PressureDrop` for each fluid.	Custom
	`HeatPump`: Identical to the heat exchanger in terms of pressure drops, but with thermodynamics of a heat pump.	Cimmino and Wetter (2017)

2.2.2 Mixing volume

A `MixingVolume` represents a fixed volume of mass that mixes perfectly with all entering fluid streams. It has a vector of fluid ports, which can have any size n. This component causes no pressure drop, hence the pressure at all its ports is the same. The main purposes of a `MixingVolume` are 1) to define a time-dependent state for the mass and energy contained within the considered volume and optionally 2) to exchange energy with a fluid stream through its heat port.

The main model equations assuming a medium with constant density and mass fractions and a state variable representing energy dynamics are

$$\sum_i \dot{m}_i(t) = 0, \tag{6}$$

$$p_i(t) = p_1(t) \qquad \forall i \in \{2, \ldots, n\}, \tag{7}$$

$$\dot{H}_i(t) = \text{semiLinear}(\dot{m}_i(t), \text{inStream}(h_i(t)), h_i(t)) \qquad \forall i \in \{1, \ldots, n\}, \tag{8}$$

$$\frac{\mathrm{d}H(t)}{\mathrm{d}t} = \dot{Q}(t) + \sum_i \dot{H}_i(t), \tag{9}$$

$$h_i(t) = \frac{H(t)}{m} \qquad \forall i \in \{1, \ldots, n\}, \tag{10}$$

$$T(t) = f_T(h_1(t)). \tag{11}$$

Equation (6) expresses conservation of mass for all mass flow rates $\dot{m}_i(t)$ flowing into port i. Since this model has no flow friction, all port pressures $p_i(t)$ are equated in Equation (7). Equation (8) computes the enthalpy flow rate $\dot{H}_i(t)$ for port i where $h_i(t)$ and $\text{inStream}(h_i(t))$ are the specific enthalpies, assuming that the fluid is exiting or entering through port i. semiLinear(, ,) is a Modelica-specific function defined as

$$\text{semiLinear}(x, y, z) = \text{if} \quad x > 0 \quad \text{then} \quad xy \quad \text{else} \quad xz. \tag{12}$$

This function either adds or subtracts energy from the mixing volume, depending on the flow direction. Equation (9) determines the time-derivative of the enthalpy $H(t)$ of the mixing volume. $\dot{Q}(t)$ represents the heat flow rate added to the volume through the heat port. The enthalpy is used in (10) to compute the specific outlet enthalpy of each port and is used in (11) to compute the heat port temperature $T(t)$ using a function $f_T(\,)$. m is the mass contained by the mixing volume.

What equations are solved for what variables depends on what components are connected to the ports of the mixing volume. A component c_1 that is connected to the heat port may for instance set the value of $T(t)$, meaning that the equations will need to be solved for $\dot{Q}$, while component c_2 may set a value for $\dot{Q}$ in which case the equations need to be solved for $T(t)$.

2.2.3 Two-way valve

The Annex 60 library version 1.0 contains models for five two-way valve types. Four of these models are very similar to the `PressureDrop` model except for how the flow coefficient is computed. The coefficient $k(t)$ now equals $g(y(t))$ where $y(t)$ is the valve control signal and $g(\,)$ is the valve characteristic such as an equal percentage valve. Each two-way valve has its own implementation for $g(\,)$. Moreover, the input signal is filtered by default, which introduces a state variable for the valve control signal. The fifth valve model, `TwoWayPressureIndependent`, uses a flow function that is pressure independent if a minimum pressure difference is available.

2.2.4 Junction

The Junction model allows three fluid flows to be connected. Each leg of the junction contains a PressureDrop component that connects it to a central MixingVolume. The Junction can be configured to have no pressure drop for its legs and the MixingVolume can be removed by setting the thermal dynamics to SteadyState. The default stream implementation (1) is then used to compute the outlet enthalpy.

2.2.5 Three-way valve

The three-way valve models are similar to the Junction models, except that the white branch in the icon has no flow friction and the black branches contain two-way valves instead of the PressureDrops. An input control signal and corresponding filter are added similar to the two-way valve model.

2.2.6 Boundary

The Boundary model sets boundary conditions for the pressure and specific enthalpy or temperature.

2.2.7 Pump

Three types of pumps exist in the Annex 60 library. An input control signal either prescribes the pressure head of the pump, the mass flow rate through the pump, or the pump speed. The pump also contains a MixingVolume that represents the thermal mass of the pump. By default a heat flow rate corresponding to the pump heat dissipation is injected in the MixingVolume.

2.2.8 Heat exchanger

This model exchanges heat between two fluid streams. Each side of the heat exchanger contains a PressureDrop. The exchanged heat flow rate is computed from the four inlet enthalpies, but only two of the inlet enthalpies are used, depending on the flow direction. When allowFlowReversal=false only the two inlet enthalpies of the design flow direction are used. The heat flow rate is used to compute the outlet enthalpies.

2.2.9 Heat pump

A heat pump is similar to a heat exchanger. The main differences are that a different function is used to compute the heat flow rate and two MixingVolumes are used to represent the condenser and evaporator. The heat flow rates are injected in these volumes.

2.3 Algebraic loops

Depending on what equations are used in components and how they are connected, algebraic loops may be formed. We first discuss a simple example, after which the solution algorithms are discussed in more detail.

Example Figure 1 shows an example model. One PressureDrop component and one two-way valve are connected to two pressure boundaries with fixed pressure such that mass flows from the high pressure boundary sou through res and val to the low pressure boundary sink. Such a model could be solved based on the following reasoning.

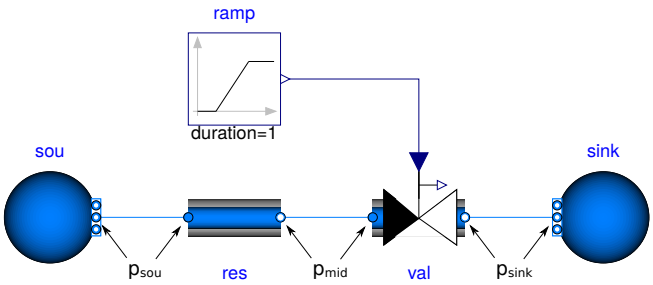


Figure 1: Example circuit with four components: two pressure boundaries, one valve and one fixed resistance pressure component.

Each port is connected to exactly one other port. Since pressures are potential variables, the pressure at the left port of `res` equals the pressure of the port of `sou`. Typically Modelica tools consider one of the variables to be an `alias` variable of the other variable. The alias variables are then eliminated from the problem, which simplifies computer algebra since fewer variables remain. Similarly, the pressure at the right port of `val` is an alias variable of the pressure of `sink` and the mass flow rates through these components are alias variables of each other. The values of the remaining pressure variable, the mass flow rate and the enthalpies need to be solved using a combination of computer algebra and numerical solvers.

The mass flow rate through this system can be computed from (2), which is contained by `res`. To compute the mass flow rate, the pressure difference over `res` needs to be known. This pressure difference equals the pressure difference between `sou` and `sink` minus the pressure drop of `val`, which in turn depends on the mass flow rate through `val`. Since mass is conserved, the mass flow rate through `val` equals the mass flow rate through `sou`, meaning that the mass flow rate through `val` needs to be known. This is however the variable that we were trying to solve for. This example illustrates that even for this simple example, an algebraic loop is formed that requires a system of equations to be solved simultaneously. The process how algebraic loops are detected and how they are solved is now explained in further detail.

Algebraic loop identi cation When translating a Modelica model, the Modelica translator collects all equations and all variables of the model. Computer algebra is then used to identify what equations depend on what variables. This information can be represented using an incidence matrix, which contains rows of equations $i = \mathit{f}1, \ldots, n\mathit{g}$ and columns of variables $j = \mathit{f}1, \ldots, n\mathit{g}$. If equation i depends on variable j, the element (i, j) contains 1, otherwise it contains 0. This information is used in Tarjan's algorithm (Tarjan, 1972) to define which equation is solved for which variable and in which sequence they are solved. The algorithm effectively restructures the incidence matrix by reordering the equations (rows) and variables (columns) such that the matrix becomes Block Lower Triangular (BLT). When there are no algebraic loops, the BLT is lower triangular. In such a matrix, the first equation e_1 uses only one variable v_1. Therefore e_1 must be solved for v_1. The second equation uses a maximum of two variables, of which one is v_1, which was solved from e_1. Therefore e_2 must be solved towards v_2 using the known value of v_1. This way all variables can be solved sequentially.

When algebraic loops exist, this sequential evaluation is not possible since there exist mutual dependencies between variables such as illustrated in the example. For each algebraic loop Tarjan's algorithm then identifies an equation set E and variable set V that need to be solved simultaneously. These sets appear as blocks on the diagonal of the BLT. Once the algebraic loop is solved, the

resulting solutions can be used to evaluate the part of the BLT that appears below the algebraic loop equations.

Algebraic loop solution Algebraic loops consist of a set of m equations E that depend on m unsolved variables in $\mathsf{V} = \{V_1(t), \ldots, V_m(t)\}$, and that may depend on state variables S that are computed by the time integrator and solved variables V that were computed earlier in the BLT. For algebraic loops the equations cannot be reordered such that they can be solved sequentially. Therefore these equation sets are solved differently, depending on the equation type and the solver implementation. In the following discussion, we assume that E contains non-linear once continuously differentiable equations, that a unique solution exists, and that the equations are solved using a Newton-type solver.

A Newton solver assumes m initial values $\{V_1^{(0)}(t), \ldots, V_m^{(0)}(t)\}$ and iteratively solves the equations. Unless the initial values are correct, the right and left hand sides of equations E return a different numerical value, implying that the equations are not satisfied. The equation residuals, i.e. differences between the left hand sides and right hand sides, and the Jacobian are typically used to compute the next values of the iteration variables. This can be repeated until a solution is found for which the residuals satisfy an error check.

This formulation requires a linear system of m equations to be solved, which may scale as $\mathcal{O}(m^3)$. Elmqvist and Otter (1994) describe *tearing*, where only a subset of V is used as iteration variables and the remaining variables are computed by solving them analytically from the iteration variables and equations E. This can significantly reduce the number of iteration variables and therefore also the computational cost.

3 Methodology

Large algebraic loops lead to longer computation times and may cause the solver to not converge. This section explains three main approaches how algebraic loops can be simplified or avoided. Firstly, algebraic loop equations can be simplified by using known **analytic solutions**. In this case, the user replaces a set of equations by a known analytical solution for these equations. Secondly, algebraic loops can be **broken**, by which we mean that they either disappear, or they are **split** into smaller parts. Thirdly, algebraic loops can be **restructured**. In this case, the structure of the equations changes. Moreover, Section 3.4 discusses how hydronic system thermal dynamics can be lumped in order to obtain faster models without introducing algebraic loops.

Jorissen, Wetter, and Helsen (2015)present the impact of these three approaches on computation time for small problems. Their work is extended in this section with a generic methodological discussion and a more complex example in Section 4 where the guidelines that were presented for these small problems cannot be applied directly. Particular attention is given to the decoupling of algebraic loops into smaller parts, which is a need that only arises in larger problems.

3.1 Using analytic solutions

In some cases, an analytic solution for the system of equations, or for a part of it, may exist. These analytic solutions are often not detected by solvers, but they can be introduced by the user.

A simple example is a series configuration of two `PressureDrop` components c_1 and c_2. Each `PressureDrop` uses nominal pressure drop parameter Δp_{nom} and nominal mass flow rate $\dot{m}_{nom}$ to compute $k = \frac{\dot{m}_{nom}}{\sqrt{\Delta p_{nom}}}$. The two `PressureDrops` can be replaced by a single `PressureDrop` with $\Delta p_{nom} = \Delta p_{nom;1} + \Delta p_{nom;2}$. The two pressure drop equations are then replaced by a single pressure

drop equation that produces the same result. Moreover, the number of variables that needs to be computed is reduced.

This can also be illustrated using the equations corresponding to Figure 1. From the connections and equations it follows that $\dot{m}_{res} = \dot{m}_{val} = \dot{m}$ and furthermore:

$$p_{sou} \quad p_{mid} = \left(\frac{\dot{m}}{k_{res}}\right)^2 \tag{13}$$

$$p_{mid} \quad p_{sink} = \left(\frac{\dot{m}}{k_{val}}\right)^2 . \tag{14}$$

Variable p_{mid} can be eliminated as

$$p_{sou} \quad p_{sink} = \left(\frac{\dot{m}}{k_{val}}\right)^2 + \left(\frac{\dot{m}}{k_{res}}\right)^2 = \dot{m}^2 \left(\frac{1}{k_{val}^2} + \frac{1}{k_{res}^2}\right) \tag{15}$$

and then solved for $\dot{m}$ as

$$\dot{m} = \sqrt{\frac{p_{sou} \quad p_{sink}}{\frac{1}{k_{val}^2} + \frac{1}{k_{res}^2}}} \tag{16}$$

to obtain an explicit analytical solution for this algebraic loop that computes $p_{sou} \quad p_{sink}$ from $\dot{m}$, or vice versa, without considering p_{mid}. These symbolic manipulations are not obvious for most solvers, but they can be integrated by the user through the use of valve parameter `dpFixed_nominal`.

These two examples illustrate that analytic solutions for series components can be used when they use the same pressure drop equation and they transition to laminar flow at the same mass flow rate. For parallel and other configurations, analytic solutions may also exist but they are harder to integrate into existing models in a user-friendly way. However, base circuit models could be developed to accommodate such functionality. Ideally Modelica tools would integrate these solutions into their solvers such that they are implemented automatically.

3.2 Algebraic loop splitting

Engineering insight sometimes allows to remove or simplify equations. Such simplifications could change certain equations in E such that they only depend on state variables S and on known variables V instead of on the algebraic loop variables V. That equation is then no longer part of the algebraic loop.

A second way to split algebraic loops is to replace one or more algebraic equations that solve for V by differential equations. The corresponding algebraic variable in V then becomes a state variable, which is computed from a differential equation. Differential equations are evaluated by the integrator, and hence are already known when the algebraic equations are evaluated. This may break algebraic loops, but it introduces additional overhead for the integrator. The introduced states typically have small time constants, which may cause smaller time steps to be required. This can negatively effect computation time, see Section 3.4.

3.3 Algebraic loop restructuring

The Annex 60 library equations are written in an explicit form where the equation is already solved for one of its variables, see e.g. (2) where Δp is evaluated explicitly from $\dot{m}$. When multiple parallel pressure drop components share the same alias variable Δp, just one distinct variable Δp can be

eliminated since the equations are not inverted by the symbolic solver. It is therefore more efficient to use its inverse, (3). Δp can then be chosen as an iteration variable, from which all mass flow rate can be evaluated using (3).

The model structure determines which of the two options is more efficient. Due to the symbolic algorithm limitations of typical Modelica tools, Annex 60 models contain a parameter `from_dp` that allows the user to switch between the two formulations manually. This can lead to a significantly more efficient evaluation of algebraic loops where a large number of pressure drop components are connected in series or in parallel.

For flow networks that combine series and parallel connections, we suggest the following rule of thumb. Whenever appropriate, series pressure drop components should be eliminated by merging them together, using parameter `dpFixed_nominal` and setting `dp_nominal=0` in the other components, which will remove the pressure drop equations. These merged series components are then connected to each other in parallel. Then set parameter `from_dp = true` to ensure that the pressure difference can be used as an iteration variable to compute the mass flow rate through each parallel branch.

For cases where the resulting algebraic loops are still too complex, using balance dynamics equations as suggested by Zimmer (2013) may offer a solution. This however requires the balance dynamics equation concept to be integrated in the Modelica standard.

3.4 Thermal dynamics configuration

The discussion thus far only concerned equations solving for pressure drops and mass flow rates. However, algebraic loops that solve for enthalpies can be formed too. For the remainder of this section, we assume all mass flow rates are known. By default, all pumps, three-way valves and junctions contain a `MixingVolume`, which introduces an enthalpy state (see (9)), due to which algebraic loops are avoided. Similarly, the control signals of pumps and valves are filtered by default, which introduces state variables. These states however have small time constants, by default in the order of 10 seconds, which can significantly increase the number of steps required by the integrator (Jorissen, Wetter, and Helsen, 2015), and is considered bad practice by Zimmer (2013). These states can be removed by setting parameter `energyDynamics` to `SteadyState`, which replaces the `MixingVolume` by an internal fluid port that acts as a mixing point (see (1)). Replacing state variables in favour of algebraic variables may however lead to linear algebraic loops, which increases the computation time for each time step. Thus there exists a trade-off between the computation time per time step and the total number of time steps.

This section discusses why these algebraic loops are formed and how they can be avoided such that the overall computation time is reduced through the use of parameters `energyDynamics` and `allowFlowReversal`.

The example at the left of Figure 2 is used to illustrate this. The subcircuit is configured using default parameter value `allowFlowReversal = true`, which leads to the variable dependencies as illustrated in Figure 2a, and using `energyDynamics = SteadyState` such that the outlet enthalpy is computed using algebraic equations as illustrated in Figure 3a. All enthalpy variables are therefore algebraic variables, which are computed from each other. Variable dependencies are indicated using arrows. Suppose now that we want to compute the heat exchanger outlet enthalpy. We then need to follow the arrows representing the equation dependencies. This shows that the variable depends on itself, since there exists a path back to the heat exchanger outlet enthalpy. Consequently an algebraic loop is generated that consists of the variables connected by red arrows in Figure 2a.

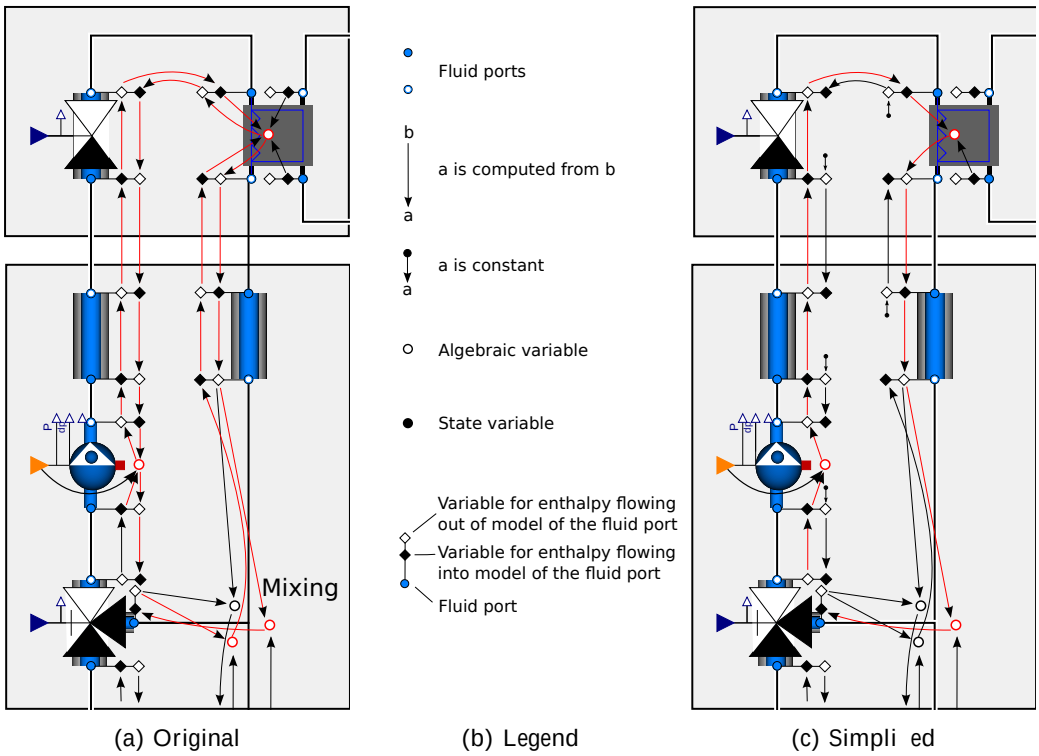


Figure 2: Equation structure for enthalpy computations of subcircuit using original (a) and simplified (c) equations. The equation structure inside of the three-way valves is illustrated by Figures 3a and 3b.

The first proposed change to remove these algebraic loops is to set `allowFlowReversal = false` in the pump, pressure drop, two-way valve and heat exchanger. This simplifies Equation (5) in each of these models. However, it should only be used if the user knows that the flow never reverses. The change results in the structure of Figure 2c. It causes enthalpy variables to depend only on the 'upstream' variables. An algebraic loop still exists as indicated in the figure using the red elements. However, it consists of fewer equations and variables. Furthermore, all equations can be solved in sequence, such that only a single iteration variable is required, since all other variables can be computed from this variable sequentially. In this particular case Dymola is even able to solve the algebraic loop analytically.

The second change consists of changing the model equations such that some of the variables that form an algebraic loop become state variables. In the previous example, one iteration variable sufficed to compute all other algebraic loop variables in sequence. Therefore, when changing a single algebraic loop variable into a state variable, all other algebraic variables can sequentially be solved from this state variable. There is however a choice where the state variable can be introduced. Typically we want to have the fewest states and the largest time constants possible. The time constants however cannot be chosen non-physically large as this would introduce too large an approximation. As a rule

(a) Steady state (b) Dynamic

Figure 3: Graphical illustration of junctions or three-way valves for a steady state (a) or dynamic (b) configuration. Annotations are added that illustrate the enthalpy equation structure. See Figure 2b for a legend of the annotations.

of thumb, we propose to introduce states at the start of a subcircuit in a model with many ports, since then all outflowing enthalpies of these ports are fixed to the state value and hence they do not end up in an algebraic loop. To avoid small time constants, the thermal mass of the state is be chosen such that it reflects the lumped thermal mass of the downstream flow leg. This way, the time constant is relatively large and the introduced state is a reasonable first-order approximation of the thermal dynamics of the subcircuit.

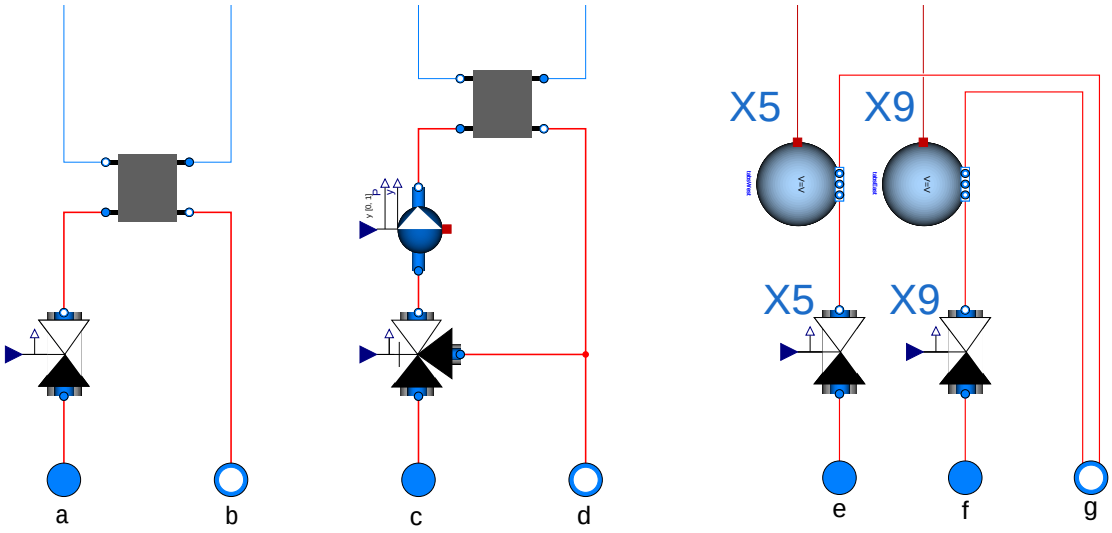
In the example, the state in the MixingVolume in the three-way valve is used. Valve output enthalpies are then directly computed from the known state values as indicated in Figure 3b. The real thermal time constant of a three-way valve is in the order of seconds and depends on the flow rate. Such small time constants may reduce the integrator time step size. Therefore a thermal mass is chosen that corresponds to the thermal mass contained by the downstream pipes.

4 Results and discussion

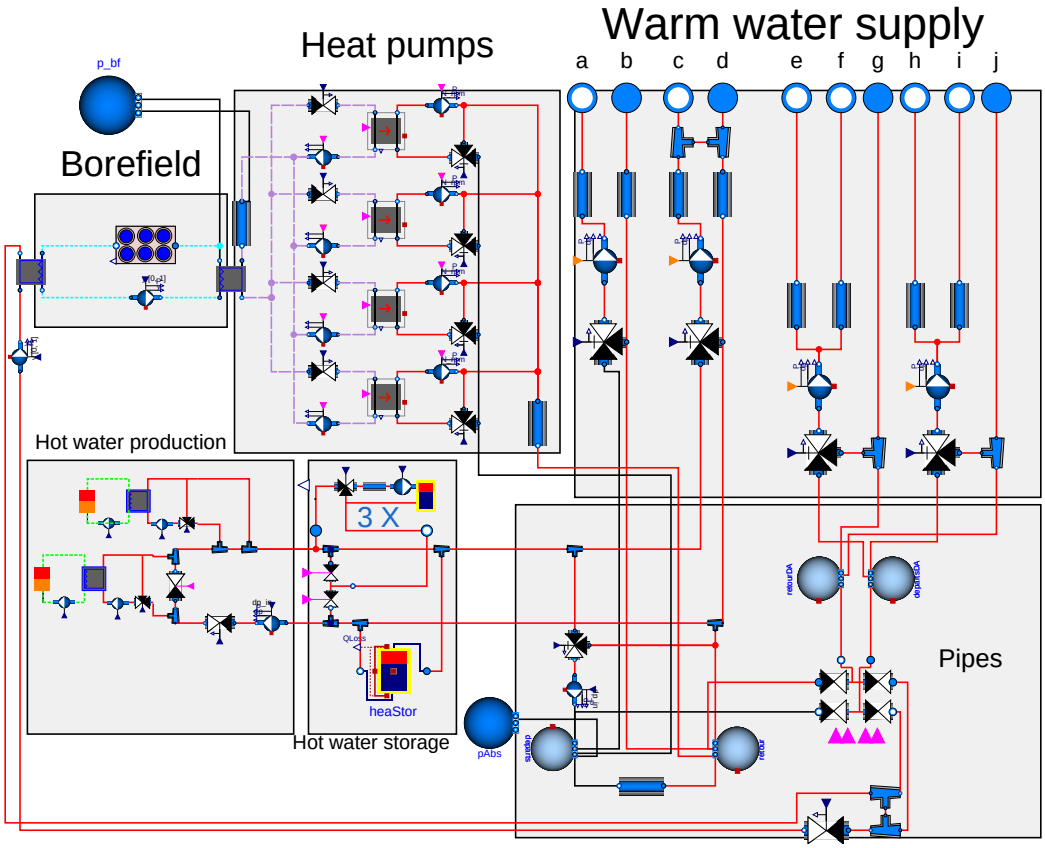
We now apply the methodology presented in Section 3 to the hydronic system of an office building model. Section 4.1 introduces the model. Section 4.2 shows how algebraic loops are split by decoupling pressure drop equations of subcircuits that interact weakly. Section 4.3 restructures algebraic loops through the use of the parameter from_dp to reduce the number of iteration variables. In Section 4.4, the remaining large algebraic loops are split up further using simplified valve models. Finally, Section 4.5 applies the above described thermal dynamics simplifications. The simplifications suggested in Section 3.1 were already applied in the initial model configuration and are therefore not discussed in detail.

4.1 Case study description

Figure 4 shows the hydronic system of a real office building. The plant is illustrated in Figure 4d. It contains a hot water section that produces hot water for the Domestic Hot Water (DHW) circuit and for the heating coils in two Air Handling Units (AHU). Hot water is stored in a hot water storage tank. Furthermore warm or cold water are supplied respectively for heating or cooling of a Concrete



(a) VAV heating coil subsystem. (b) AHU heating coil subsystem. (c) South CCA subsystem.



(d) Main hydronic network model consisting of cold, warm and hot water sections.

Figure 4: Modelica model of the hydronic system. Lines between components represent connections. Lines with the same color and line type indicate that equations corresponding to the linked components form one algebraic loop. Black connections indicate that the pressure equals the fixed boundary pressure. The subsystem shown in subfigure (c) and 24 parallel instances of subfigure (a) and (b) are connected to subfigure (d) using the indicated ports. A second instance of subfigure (c) is connected to ports h, i, j of subfigure (d). One large algebraic loop is formed, indicated in red, and multiple smaller algebraic loops.

Core Activation (CCA) ceiling heating or cooling system. Warm water is produced using four heat pumps and is also used as a heat source for 24 Variable Air Volume (VAV) heating coils. Heat or cold for the heat pumps and passive cooling is supplied by a borefield model at the top left. The subsystem at the top of the hot water storage box consists of three parallel subsystems that each charge one DHW storage tank using a three-way valve and a pump. Furthermore ports indicated using letters 'a' through 'j' are external connections to more components outside of Figure 4d:

24 instances of the VAV heating coil subsystem in Figure 4a are connected to ports 'a' and 'b' in parallel.

2 instances of the AHU heating coil subsystem in Figure 4b are connected to ports 'c' and 'd' in parallel.

The CCA subsystem in Figure 4c supplies water to the CCA circuits. 5 CCA instances are connected to supply ports 'e' and 9 to port 'f'. Port 'g' is a shared return flow port.

5 CCA instances are connected to supply ports 'h' and 10 to port 'i'. Port 'j' is a shared return flow port.

Unless carefully constructed, a model like this contains large non-linear algebraic loops. We denote the size of an algebraic loop by `(x,y)`, where x is the number of *equations* in the algebraic loop and y is the number of *iteration variables*. A partial list of non-linear algebraic loops generated when using Dymola version 2017FD01 for the model corresponding to Figure 4d is `[(259, 86), (3, 1), (3, 1), (22, 4), (7, 2)]`. This does not include algebraic loops related to enthalpy computations, which are discussed in Section 4.5.

Each of these algebraic loops correspond to a specific part of the hydronic circuit. In Figure 4, different colors are used to indicate the subcircuits that form one algebraic loop. The two algebraic loops with sizes `(3, 1)` compute the mass flow rates and pressures of the two fluid loops indicated in green in the two heat production subcircuits at the bottom left of the figure. The algebraic loop with sizes `(7, 2)` solves the blue fluid loop of the borefield. The algebraic loop with sizes `(22, 4)` solves the purple algebraic loop at the primary side of the heat pumps. Most remaining components are coupled into one large (red) algebraic loop, which has sizes `(259, 86)`. This algebraic loop cannot be solved by the Newton solver, i.e. the solver does not converge and the simulation is aborted.

4.2 Algebraic loop splitting

Large algebraic loops in flow networks can often be split into smaller parts by decoupling weakly coupled subcircuits, i.e. circuits where mass flow rates and pressure drops in one part of the circuit only have a small influence on the mass flow rates and pressure drops in the other parts. These weak couplings are often not of interest for building energy simulations and can therefore be removed. Algebraic loops spanning over two subcircuits can typically be decoupled by removing pressure drop equations that require mass flow rates and/or pressure drops of both subcircuits to be known as follows.

Consider for instance the two `MixingVolumes` (see Table 1) at the bottom of Figure 4d. They represent a supply (left) and return (right) water collector of the heat pumps. A bypass between these collectors compensates mass flow rate imbalances that exist between the heat pump supply pumps and the warm water demand pumps shown in Figure 4a and Figure 4c. The purpose of this bypass is therefore to decouple subcircuits such that the pumps operate independently from each other. This bypass was however modelled using a pipe, which causes a pressure drop. The pressure drop Δp_b across this bypass can be computed from the difference of all mass flow rates flowing to

and from both collectors. These mass flow rates are however a function of the pressure difference between the collectors, which are in turn functions of Δp_b. Hence an algebraic loop is generated, consisting of the equations from supply and return connections from and to the collectors.

This algebraic loop is split by removing the pressure drop over the bypass through setting `dp_nominal = 0`. Since each collector then has the same pressure, each subcircuit coupled to the collectors can compute the mass flow rates using the known pressure difference of 0 Pa across the bypass.

A similar bypass consists of the connection made by the storage tank at the bottom of the 'Hot water storage' block in Figure 4d. Three subcircuits are connected in parallel to this storage tank. The first is the hot water production circuit on the left. The second is the circuit connecting to ports (c) and (d). The third is the circuit connecting the tank to the heat pump collectors in the bottom center of the figure. The goal of this system design is to allow each of these subcircuits to inject or withdraw hot water from the tank independently. Therefore we assume that the pipes are sized sufficiently large such that the mutual influence created by the pressure drops is negligible. Furthermore a pressure independent valve is used to control the mass flow rate in the heat production circuit. This valve contains an internal control loop that tracks the flow rate set point such that the flow rate is not influenced by the pressure difference across the valve provided sufficient pressure is available. The second subcircuit is connected to long pipes. The pressure drop of these pipes is significantly larger than the pressure drop over the short pipes towards the heat storage tank. The connection to the warm water collector is not used in practice and therefore its mass flow rate is always zero. Therefore, the pressure drop across the heat storage tank can be neglected, since it will not significantly influence the mass flow rates in the circuits.

More bypasses are formed by the two `Junctions` at:

the bottom right in Figure 4d,

ports (c) and (d),

the subcircuit at the top in the 'Hot water production' part.

The pressure drops generated by these bypasses were also removed.

The result of these simplifications can be seen by comparing Figure 4 and Figure 5. Black lines in both figures indicate connections for which the pressure along the line is fixed and equal to the boundary condition pressure set by component `pAbs`. After the simplifications a significant part of the system is at the fixed boundary pressure. This causes the algebraic loops to be split up into smaller parts. The original algebraic loop with size `(259, 86)` is split into smaller algebraic loops with sizes `[(65, 28), (55, 25), (34, 5), (21, 4), (12, 2), (7, 1), (7, 1), (7, 1), (6, 1), (3, 1)]`. Each set of red connections in Figure 5 that is separated from other components by black connections represents one algebraic loop.

After these simplifications most algebraic loops have only a few iteration variables. However, two algebraic loops with 28 and 25 iteration variables remain. In the next section these algebraic loops are restructured such that the number of iteration variables is reduced further.

4.3 Algebraic loop restructuring

Consider the algebraic loop with sizes (55, 25). It computes the mass flow rates through 24 valves that are connected in parallel to ports (a) and (b) (see Figure 4a). Each valve contains a pressure drop equation, which by default uses `from_dp = false` such that (2) is used. In order to compute

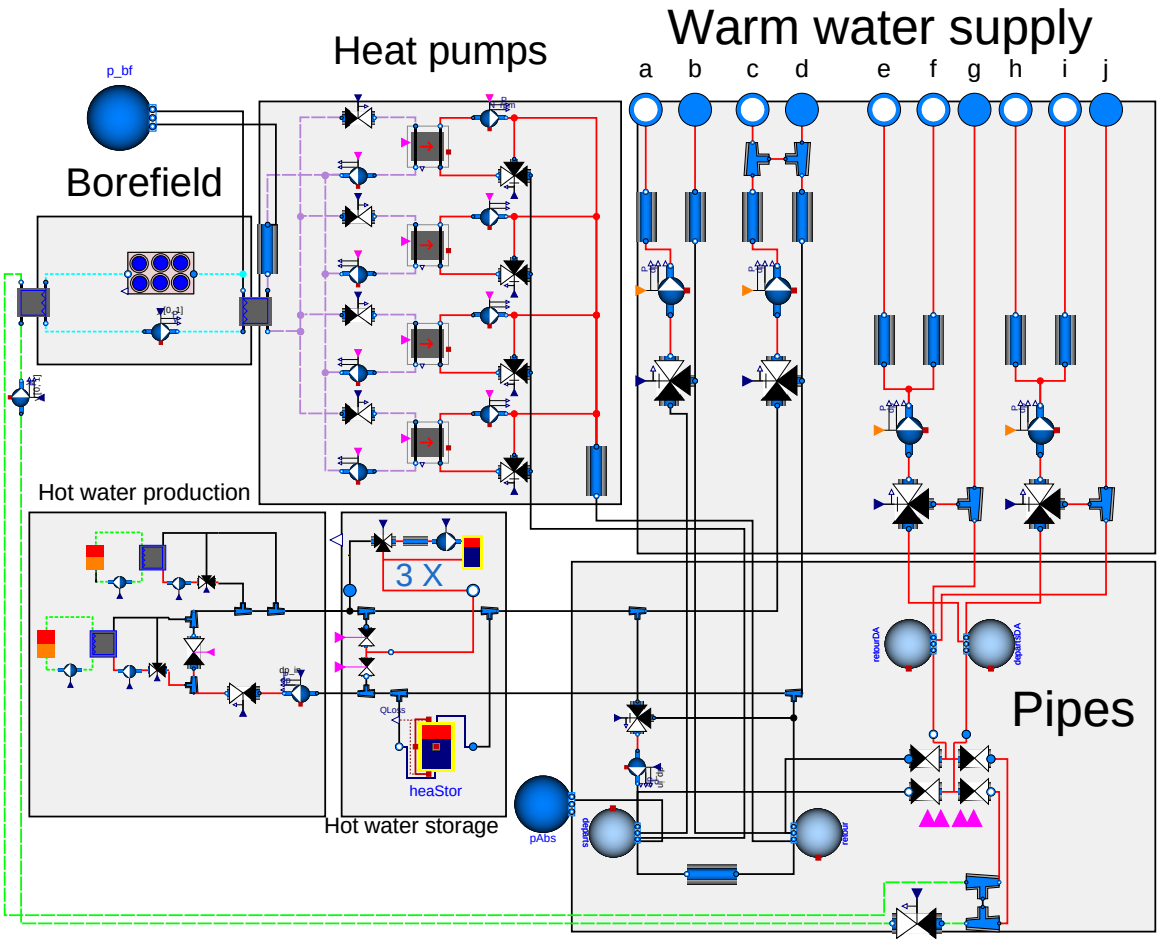


Figure 5: Illustration of algebraic loops when bypasses cause no pressure drops. The large algebraic loop is split into smaller parts.

the 24 mass flow rates through the 24 parallel branches, each mass flow rate must either be an iteration variable of the algebraic loop, or it must be solved from an equation using the iteration variables. The algebraic loop contains 26 equations in which the 24 $\dot{m}_i$ variables appear:

24 instances of (2): 1 for each valve,

conservation of mass at port (a): $\sum \dot{m}_i = \dot{m}_a$,

conservation of mass at port (b): $\sum \dot{m}_i = \dot{m}_b$.

Equation (2) cannot be solved analytically for $\dot{m}$ unless the function is symbolically inverted by the solver, which does happen in Dymola for the analysed cases. The other two equations are linear equations that may be solved symbolically for the mass flow rates. This means that only 2 equations can be solved towards mass flow rates. This is 22 equations too few, so at least 22 mass flow rates need to be iteration variables. Due to the additional equations introduced by the pipes, valve and pump, the total number of iteration variables is 25: 23 instances of $\dot{m}_i$ and the mass flow rate

through ports (a) and (b).

As explained in Section 3.3 the problem structure forces the solver to use many iteration variables, which can be overcome by setting `from_dp = true` in the valve models. The solver should then be able to identify that all valves have the same pressure drop and hence a single iteration variable can be used. However, Dymola 2017FD01 does not identify that the pressure drops are alias variables and therefore they are not eliminated. Hence 24 iteration variables are required. Further simplifications of the flow network are required before the alias variables are identified. In this case the `PressureDrop` components that are connected to ports (a) and (b) can be merged into one `PressureDrop` component since the same mass flow rate flows through them. When the `PressureDrops` are merged, a solution is found where only a single iteration variable is required: the pressure at port (a). Using this iteration variable, the mass flow rates through all valves can be computed. This determines the total mass flow rate through the left `PressureDrop` component, from which its pressure drop is computed since this component by default has `from_dp`=false. The pump prescribes the pressure difference across the pump such that the pressure at its inlet may be computed from this set point and the computed pressure drop. Finally that pressure may be used to compute the mass flow rates through the two branches of the three-way valve, which by default have `from_dp`=true. The residual equation equals the sum of the mass flow rates of the three-way valve ports.

Similarly, `from_dp`=true was set for the valves in the algebraic loop of sizes `(65, 28)`. The resulting algebraic loop has 7 iteration variables instead of 28. However, it can still not be solved robustly. The following subsection explains how this algebraic loop, and two others are simplified further.

4.4 Valve simplifications

Below, three subcircuits containing valves are simplified by removing their pressure drop computations, which will further split up algebraic loops. These specific valves can be simplified because their function is not to control the flow magnitude by causing flow friction, but rather to direct the entire flow towards one of two possible paths, which can be simulated without using pressure drop equations.

4.4.1 Valve simplification 1

The first considered subcircuit in Figure 5 is the bottom circuit indicated in red at the bottom of the 'Hot water production' block. The size of this algebraic loop is `(12, 2)`. Further simplification is not required since the number of iteration variables is already small, but we demonstrate how this could be achieved since the same reasoning can be applied to larger circuits.

The left-most two-way valve in this circuit is a valve that is either fully open or fully closed. Since the pressure drop of the open valve is negligible, it is safe to assume the pressure drop to be zero when the valve is open. The valve should be able to block the flow by creating a pressure difference when the valve is closed. However, the building controller is configured such that this valve is only closed when the pump is off. Therefore this valve can be replaced by a dummy valve that causes no pressure drop in both positions, since the system already ensures that the flow is zero when the valve is closed. This decouples the two parts of the subcircuit. The dummy valve does however verify our assumption: whether the mass flow rate through the valve is indeed zero when the control signal indicates that the valve is closed. This avoids unintended modelling errors.

4.4.2 Valve simplification 2

A second simplification can be made for the two valves in the middle of the 'Hot water storage' block, which has sizes `(21, 4)`. These valves determine whether the return water from the three storage tanks is injected in the top pipe, or in the bottom pipe. The building controller activates the circulation pumps only when one of the two valves is fully open. The flow then exits through one of the two valves. The valve control signal is sufficient to know what fraction of the flow exits through each valve. Therefore there is no need to use pressure drop equations to compute this fraction. The model can therefore be simplified by neglecting the pressure drop equations and by enforcing the flow rate fractions explicitly. A second dummy model was created that achieves this, using

$$p_c = p_b, \tag{17}$$
$$0 = \dot{m}_a + \dot{m}_b + \dot{m}_c, \tag{18}$$
$$\dot{m}_a = \text{ if } u_1 \text{ then } \quad \dot{m}_c \text{ else } 0, \tag{19}$$
$$h_a = \text{inStream}(h_c), \tag{20}$$
$$h_b = \text{inStream}(h_c), \tag{21}$$
$$h_c = \text{if } u_1 \text{ then inStream}(h_a) \text{ else inStream}(h_b), \tag{22}$$

where u_1 and u_2 are two boolean control signals and h_a, h_b, h_c, inStream(h_a), inStream(h_b), inStream(h_c) and $\dot{m}_a$, $\dot{m}_b$, $\dot{m}_c$ are specific enthalpies and flow rates corresponding to fluid ports (a), (b), (c). Port (a) and port (b) are respectively connected to the bottom and top pipe and port (c) is connected to the domestic hot water storage tanks.

This model also verifies whether the mass flow rates are zero when both valves are closed such that $u_1 = u_2 = 0$. Furthermore this implementation is only correct when the pressures at ports (a) and (b) are equal since it always assumes $p_c = p_b$. This is true in our case since the parallel storage tank model does not have a pressure drop. This simplification therefore requires knowledge about the interaction between system components, which defeats the object orientation paradigm.

After the simplification, this algebraic loop is split up into 3 smaller algebraic loops, one for each parallel branch in the DHW circuit. They each have one iteration variable.

4.4.3 Valve simplification 3

At the right of Figure 6 one large algebraic loop remains, which has 7 iteration variables. It computes the flow rates through the north and south CCA circuits, which are coupled through the four valves at the bottom of the circuit. The valves open a connection to either the passive cooling circuit or the heating circuit. Only one of the two options is opened at any given time and the pumps are only enabled when either of the two sets of valves is opened. Therefore the same reasoning as in the second valve simplification is applied, now requiring one instance of the second simplified valve model and a second instance of this model where (17) - (19) are replaced by

$$p_c = p_b, \tag{23}$$
$$p_a = p_b, \tag{24}$$
$$\dot{m}_b = \text{ if } u_1 \text{ then } 0 \text{ else } \quad \dot{m}_c. \tag{25}$$

This ensures that the total set of equations is non-singular.

The resulting set of non-linear algebraic loops is now `[(54, 1), (34, 5), (24, 3), (23, 3), (7, 1), (7, 1), (7, 1), (7, 1), (6, 1), (6, 1), (6, 1), (5, 1), (5, 1)]`. Figure

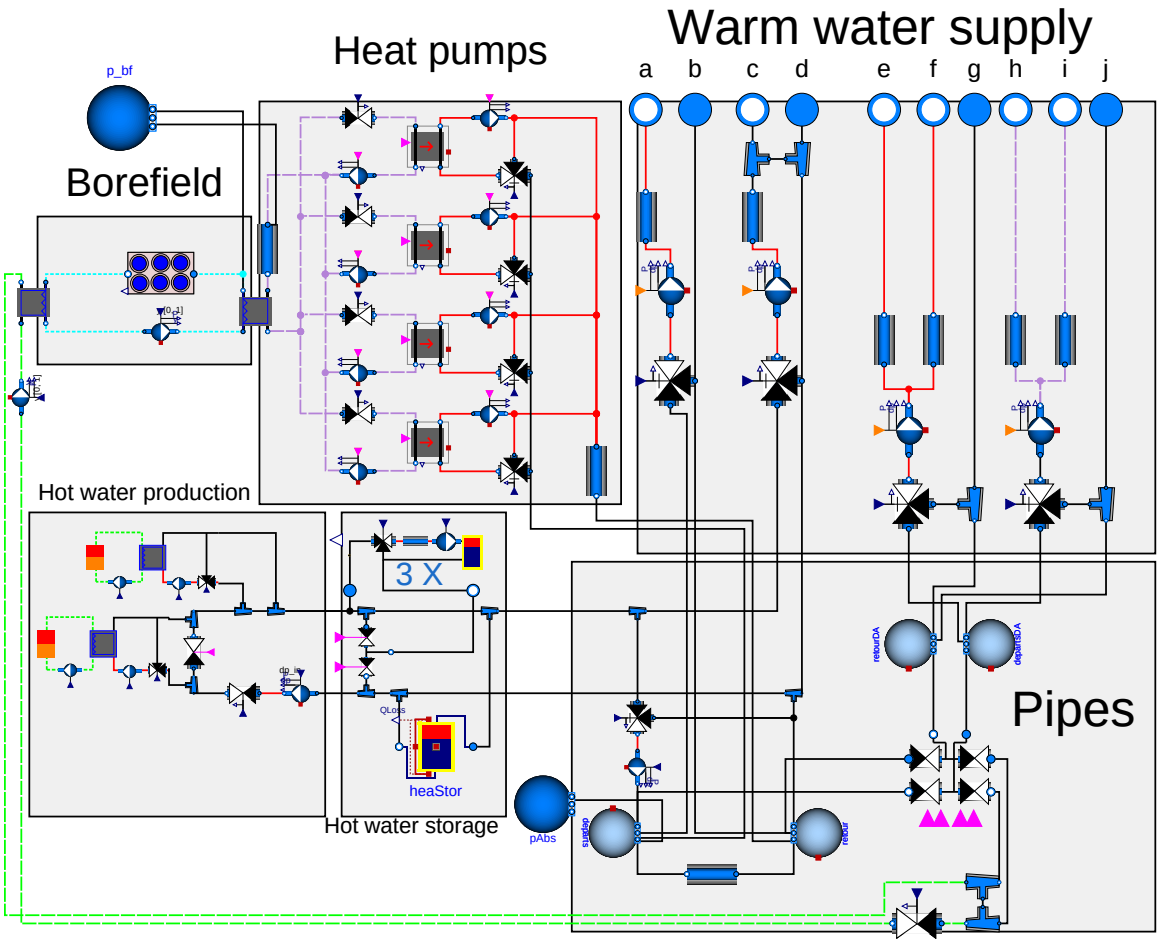


Figure 6: Figure showing that algebraic loops are broken up into smaller parts when the certain valves are simplified.

6 provides a graphical illustration of the final set of algebraic loops. Only small algebraic loops remain. The largest remaining algebraic loop is the algebraic loop at the secondary side of the heat pumps. It has size (34, 5). Further simplification can be achieved by removing the fixed pressure drop element of the common pipes, but measurement data has shown that this would lead to too large an error since its pressure drop is not negligible.

4.5 Thermal dynamics

This section discusses how thermal dynamics were configured to avoid algebraic loops, while also avoiding small time constants. As a reference configuration, all states were removed except those of heat storage devices, heat production devices and the borefield. An algebraic loop is then generated with sizes `(632,123)`.

As suggested in Section 3.4 all models were configured with `allowFlowReversal = false`. This resulted in a reduced algebraic loop of size `(435, 54)` due to the simplification of (5).

Thermal dynamics were then introduced for all three-way valves and junctions at the start of each subcircuit and in mixing volumes that represent thermal collectors. This removed algebraic loops since all variables can be computed sequentially from the new state variables.

The overall result of these simplifications is a model of which the smallest time constant is on the order of one minute and with algebraic loops containing up to five iteration variables. The smaller algebraic loops reduce the computation time per time step and the model time constants allow the use of explicit Euler integration. This is a relatively fast time integrator for problems with many state variables (Jorissen, Wetter, and Helsen, 2015). After all these modifications, the hydronic system, when coupled to a building envelope model, ventilation systems and a controller model, can be simulated 100 times faster using Euler integration than with Dymola's default integration algorithm, Dassl. As the original model did not converge, the simplifications and methodology presented in this work were crucial to obtain a fast implementation of a detailed building model in Modelica.

5 Conclusion

Modelica is a multi-disciplinary modelling language that is well suited for modelling buildings and their HVAC systems using a relatively high level of detail. However, when unaware of the algorithms used by Modelica tools, for large systems, the Modelica user can easily construct models that are slow or models for which the solver does not converge. One of the primary causes for this is the generation of algebraic loops, in particular those related to flow networks. The main contribution of this paper is that it explains why the model equations of hydronic flow networks lead to algebraic loops and how these algebraic loops can be simplified or avoided altogether. This knowledge can be applied directly by Modelica users, or it can be used by library developers, e.g. to develop preconfigured subcircuit templates.

We propose a methodology to reduce the size of algebraic loops, which is an application of basic methods proposed by Jorissen, Wetter, and Helsen (2015). Firstly, analytic solutions should be used for problems whenever they are available. Secondly, algebraic loops can be split by simplifying negligible model equations. Thirdly, algebraic loops can be restructured such that they are solved using fewer iteration variables. Furthermore, thermal dynamics can be simplified by configuring models to only allow one flow direction and by lumping the thermal mass of subcircuits at the start of the circuit. Some of these recommendations implicitly hold simplifications, such as: some pressure drops are negligible relative to others, flow reversal is not taken into account, fast dynamics of the HVAC are negligible.

This methodology is applied to an example hydronic system of an office building. Pressure drop equations in this flow network lead to an algebraic loop with 86 iteration variables. These equations were simplified using our methodology, which leads to algebraic loops with a maximum of 5 iteration variables. Thermal equations in the flow network with quasi-stationary thermal dynamics lead to an algebraic loop of 123 iteration variables. Our methodology removed the algebraic loop completely. When combined with a suitable integration algorithm, this can lead to models that are two to three orders of magnitude faster than non-optimised models.

Some of the proposed simplifications require knowledge of the modelled system and are therefore not easily applied. Further research and development of specialised subcircuits, e.g. within the scope of IBPSA project 1, can better expose these modelling options to users. For an open-source, documented example application based on this paper see the IBPSA library package `IBPSA.Fluid.Examples.FlowSystem`.

Acknowledgements

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 developed and demonstrated new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

We thank Boydens engineering and in particular Wim Boydens and Kurt Corvers for their collaboration, which made possible the development of the presented case study.

Funding

This work was supported by the Agency for Innovation by Science and Technology in Flanders (IWT); under Grant 131012 and 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.

References

Baetens, Ruben, Roel De Coninck, Filip Jorissen, Damien Picard, Lieve Helsen, and Dirk Saelens. 2015. "OPENIDEAS - An Open Framework for Integrated District Energy Simulations." In *14th Conference of International Building Performance Simulation Association*, Hyderabad. International Building Performance Simulation Association.

Cimmino, Massimo, and Michael Wetter. 2017. "Modelling of Heat Pumps with Calibrated Parameters Based on Manufacturer Data." In *12th International Modelica Conference*, edited by Jiří Kofránek and Francesco Casella, Prague, 219–226. Modelica Association and Linkŏping University Electronic Press.

Elmqvist, Hilding, and Martin Otter. 1994. "Methods for tearing systems of equations in object-oriented modeling." In *Proceedings of the European Simulation Multiconference*, Vol. 2, 326–332.

Franke, Rüdiger, Francesco Casella, Martin Otter, Michael Sielemann, Hilding Elmqvist, Sven Erik Mattson, and Hans Olsson. 2009. "Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena." In *Proceedings 7th Modelica Conference*, Como, oct, 108–121.

Jorissen, Filip, Michael Wetter, and Lieve Helsen. 2015. "Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation." In *11th International Modelica Conference*, Paris, France, 59–69.

Mattsson, Sven Erik, Hilding Elmqvist, and Jan F. Broenink. 1997. "Modelica-An international effort to design the next generation modeling language." *Journal A, Benelux Quarterly Journal on Automatic Control* 38 (3): 16–19.

Müller, D., M. Lauster, A. Constantin, M. Fuchs, and P. Remmen. 2016. "AIXLIB – An Open-Source Modelica Library Within the IEA-EBC Annex 60 Framework." In *BauSIM*, edited by John Grunewald, Clemens Felsmann, Andreas Nicolai, and Joachim Seifert, Dresden, 3–9.

Nytsch-Geusen, Christoph, Jörg Huber, Manuel Ljubijankic, and Jörg Rädler. 2013. "Modelica BuildingSystems- eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme." *Bauphysik* 35 (1): 21–29.

Tarjan, Robert. 1972. "Depth-first search and linear graph algorithms." *SIAM Journal on Computing* 1 (2): 146–160, doi:10.1137/0201010.

U.S. Department of Energy. 2016. *EnergyPlus Version 8.7 Documentation Engineering Reference*.

Wetter, Michael, Marcus Fuchs, Pavel Grozman, Lieve Helsen, Filip Jorissen, Moritz Lauster, Müller Dirk, et al. 2015. "IEA EBC Annex 60 Modelica Library - An International Collaboration to Develop a Free Open-Source Model Library for Buildings and Community Energy Systems." In *14th Conference of International Building Performance Simulation Association*, edited by Jyotirmay Mathur and Vishal Garg, Hyderabad, 395–402. International Building Performance Simulation Association.

Wetter, Michael, and Christoph van Treeck. 2017. *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*.

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

Zimmer, Dirk. 2013. "Using Artificial States in Modeling Dynamic Systems: Turning Malpractice into Good Practice." In *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, edited by Henrik Nilsson, Nottingham, 77–85. Linköping University Electronic Press; Linköpings universitet.