Journal of Building Performance Simulation
Vol. 2, No. 2, June 2009, 143–161

Modelica-based modelling and simulation to support research and development in building energy and control systems

Michael Wetter*

Lawrence Berkeley National Laboratory, Energy and Environmental Technologies Division, Building Technologies Program, Berkeley, CA, USA

(*Received 1 December 2008; final version received 12 February 2009*)

Traditional building simulation programs possess attributes that make them difficult to use for the design and analysis of building energy and control systems and for the support of model-based research and development of systems that may not already be implemented in these programs. This article presents characteristic features of such applications, and it shows how equation-based object-oriented modelling can meet requirements that arise in such applications. Next, the implementation of an open-source component model library for building energy systems is presented. The library has been developed using the equation-based object-oriented Modelica modelling language. Technical challenges of modelling and simulating such systems are discussed. Research needs are presented to make this technology accessible to user groups that have more stringent requirements with respect to the numerical robustness of simulation than a research community may have. Two examples are presented in which models from the here described library were used. The first example describes the design of a controller for a nonlinear model of a heating coil using model reduction and frequency domain analysis. The second example describes the tuning of control parameters for a static pressure reset controller of a variable air volume flow system. The tuning has been done by solving a non-convex optimization problem that minimizes fan energy subject to state constraints.

Keywords: building simulation; equation-based modelling; rapid prototyping; Modelica; controls

1. Introduction

To design and operate energy efficient buildings, it is important to properly account for the dynamic system performance over a wide range of time scales and operating conditions. At the time scale of hours to days, the system dynamics determine how much energy can be stored passively or actively to exploit natural sources for heating and cooling. At the time scale of minutes or even seconds, the system dynamics determine whether equipment is cycling, which can negatively impact its energy performance and may cause premature equipment failure. Also, the performance of certain equipment can peak at part load and degrade at low and high load. To design control algorithms that exploit such behaviour in order to increase system-level efficiency, simulation models need to reflect the change in efficiency at various steady-state and dynamic operating conditions. Temporally averaging the performance of cycling equipment, as is customary in hourly building simulation programs, is not always satisfactory as it does not account for the high cycling frequency which can degrade the equipment performance. An example is a direct evaporating cooling coil (DX coil) where the condensate may evaporate into the supply air when the compressor is switched off (Henderson and Rengarajan 1996).

In industrial research, model-based system engineering is increasingly used to reduce product development cycles and to fix errors early in the design. Such a process often involves experts from different disciplines who need to integrate models of various domains to analyse their interaction or to redesign components in a way that increases system-level efficiency or controllability (Banaszuk *et al*. 2007). Such processes require models that can be used beyond time-domain simulation, for example in conjunction with frequency domain analysis (to analyse stability and design beneficial dynamic behaviour) or with optimization algorithms (to optimize the design or operation).

Using building simulation programs for such applications leads to new requirements for modelling and simulation tools. For example, to support the invention of new systems for space conditioning, building simulation programs need to allow a scientist to quickly add new models and use the models within rapid prototyping processes. For multidisciplinary research, it needs to be possible to integrate different models that have been developed concurrently by different domain experts who are part of a

*Email: mwetter@lbl.gov

ISSN 1940-1493 print/ISSN 1940-1507 online

DOI: 10.1080/19401490902818259
http://www.informaworld.com

multidisciplinary research team. For controls design and analysis, building simulation programs need to represent the dynamic behaviour of components and their interaction within a system. For generating a simulation model of an HVAC system and its controls from a Building Information Model, building simulation programs should be able to represent any HVAC and controls configuration that can be built in reality. This would allow using any valid Building Information Model that contains not only building envelope data but that may in the future also contain data for HVAC components, their system configuration and the specification of control algorithms.

We recognize that vendors may not release the performance data and control algorithms at a level of detail needed for a proper simulation of the dynamic system performance. In such situations, one can still use the pragmatic approach of implementing simplified equipment performance curves and control algorithms with equation-based modelling languages, using the scarce information available, as is frequently done in today's building simulation programs.

2. Traditional building simulation programs

To better distinguish the modelling and simulation technique described in this article from the way that building simulation programs are typically written, we introduce the term *traditional building simulation programs*. By traditional building simulation programs, we mean building simulation programs that are written using an imperative language, such as FORTRAN, C and C++. Examples of traditional building simulation programs include DOE-2, ESP-r and EnergyPlus (Winkelmann *et al.* 1993, Clarke 2001, Crawley *et al.* 2001). In such programs, a developer writes sequences of computer instructions that assign values to variables in a predefined order of execution. Typically, such programs mix code that describes the physical process with code for data management and for numerical solution methods.

Traditional building simulation programs have in general not been designed based on the above requirements. Meeting these requirements requires modelling techniques that are different from the techniques used in these programs, in which the semantic gap between simulation model and actual component can be large and which often mix code for expressing the physical behaviour with code for data management and numerical solution methods. A reason for this semantic gap is that models in traditional building simulation programs are written in a way that was motivated by how computers process instructions. For example, when implementing a physical model, a program developer sorts and manipulates the physical equations based on what variables are known (the input) and what variables need to be computed (the output). Then, the program developer writes causal, sorted variable assignments and implements them in a source code. This source code may call other program procedures, thereby transferring the locus of control until the response of a subsystem to its input variables is computed. During this process, program procedures may set flags to request from the solver the re-simulation of a subsystem. Clearly, this is not how one would state the physical laws that govern constraints between component interface variables and how one would describe how components interact with each other. Rather, it was the only approach for writing building simulation programs at a time when more modern tools for modelling, symbolic algebra, numerical solution and code generation were in their infancy.

3. Limitations of traditional building simulation programs

In traditional building simulation programs, component models frequently integrate their own numerical solver and mix program flow logic with equations that simulate the physical behaviour. This leads to a program code that is hard to maintain and for which it is difficult to add new models. The nested solvers can also lead to significant numerical noise in the simulation results that can make the use of optimization programs difficult (Wetter and Wright 2004, Wetter and Polak 2004b). Furthermore, the lack of separation between models, data and solvers makes it hard to integrate models from different disciplines for co-simulation such as in Trcka *et al.* (2006, 2007) and Wetter and Haves (2008). Because the majority of building simulation programs do not model the dynamics of HVAC systems and sometimes implement an idealized controller directly in a component model, many standard control sequences such as the ones described by ASHRAE (2005) and CIBSE (2000) are difficult if not impossible to model. For example, in EnergyPlus, fan coil units are controlled based on the zone heating and cooling load and not based on a zone thermostat. This makes it hard to use such models in conjunction with models of control systems.

As part load performance and system controls become of increasing importance in very low energy buildings, energy simulations need to properly resolve the nonlinear dynamic behaviour of building energy and their control systems, and capture the dominant dynamics that may lead to equipment short-cycling. When developing models for such systems, one obtains systems of differential equations where the time constants can vary significantly among different components. These systems of differential equations may be coupled to algebraic equations and to difference equations. For the efficient numerical solution of such systems, algorithms for symbolic computer algebra (such as partitioning and tearing) and implicit numerical solvers for differential equations should be used (Cellier and Kofman 2006).

With respect to composing HVAC systems from component models, we note that in some traditional building simulation programs, the arrangement of HVAC and control components is governed by composition rules (that define how components can be assembled to form a system) which are not devised based on how actual HVAC components can be connected to form a system. Rather, the composition rules were defined such that an efficient numerical solution could be obtained in a program architecture that distributes solvers to individual components and subsystems and that does not make use of symbolic manipulations. This led to program architectures with sequential simulation of loads, systems and plants, or to program architectures that are based on fluid loops. The consequence is that translating an HVAC system from a Building Information Model to an energy simulation program is only possible if the HVAC system instantiation in the BIM has been done such that the component connectivity conforms to the composition rules of the energy simulation program. This is, for example, followed in the approach described by Bazjanac and Maile (2004). Although this approach may present a working solution for traditional HVAC systems and controls, it presents a challenge for the use of BIM for non-standard HVAC systems and to represent novel control algorithms.

4. Building system library

Because of these limitations, we started an open-source development of a new component library for building energy systems using the equation-based object-oriented modelling language Modelica (Mattsson and Elmqvist 1997). For early applications, we are primarily interested in:

- offering innovative companies and researchers a platform to stimulate innovations in energy-efficient building systems;
- enabling virtual rapid prototyping to evaluate different concepts rapidly so that promising alternatives can be identified for further refinement and product development;
- enabling researchers to quickly add models of emerging technologies into the simulation environment to do performance assessment; and
- enabling controls engineers to extract different subsystem models from models that are used for the design of the building energy system. These models may then be used within a feedback controls design process, or they may be embedded within building control systems for model-based controls, fault detection and diagnostics.

In the long term, we envision the models being used in a work flow that automatically generates a simulation model from a BIM that includes a modular definition of HVAC systems and control algorithms.

Modelica has been applied earlier for building energy modelling applications but an extensive library with both steady-state and dynamic component models for building energy systems is not yet available. Other Modelica development has been reported by Merz (2002) and Felgner *et al.* (2002) who describe the library ATPlus for thermal building simulation. Hoh *et al.* (2005) expanded the components of the ATPlus library to include a room model with heat exchangers that are embedded in wall constructions. Nytsch-Geusen *et al.* (2005) developed a hygrothermal building model as part of a Modelica library for multizone building heat and mass transfer analysis. A multi-zone thermal building model is described in Wetter (2006b) and a multizone airflow model is described in Wetter (2006a).

4.1. *Characteristics of the target applications*

To further illustrate why we are interested in a new approach for modelling and simulation of building systems, and to rationalize our selected implementation, we will now discuss some characteristics of the above applications:

4.1.1. Efficient numerical solution

To discuss the computation time, consider the equation

$$\frac{\text{seconds}}{\text{program}} = \left(\frac{\text{instructions}}{\text{program}}\right) \left(\frac{\text{clocks}}{\text{instruction}}\right) \left(\frac{\text{seconds}}{\text{clocks}}\right) \tag{1}$$

that describes the time needed to execute a program as the product of the number of instructions of the program, times the average clock cycles needed to process an instruction, times the seconds that elapse per clock cycle. As application developers, we can influence the first two terms on the right-hand side, whereas the third is given by the processor clock.

The instructions to be executed by the program can be reduced by selecting efficient symbolic and numerical algorithms and a good software structure. With regard to numerical algorithms, we note that in building energy systems, time constants vary between seconds for feedback control to days for energy storage. To ensure stability, to control the error of the numerical solution and for computational efficiency, solving such systems requires implicit integration algorithms with adaptive step sizes (Hairer and Wanner 1996, Cellier and Kofman 2006). This, in turn, can lead to large linear and nonlinear systems of equations that need to be solved simultaneously. Since the computation time of linear and nonlinear solvers is typically proportional to n^2 to n^3, where n denotes the number of unknowns that are solved for simultaneously, it is generally advantageous to reduce n. For equation-based languages, this can be done by symbolically manipulating the system of equations using methods such as partitioning, tearing and inline integration (Elmqvist *et al.* 1995, Bunus and Fritzson 2004, Cellier and Kofman 2006), which are similar to the methods used in SPARK (Sowell *et al.* 1986, Sowell and Haves 2001, Wetter *et al.* 2008). However, traditional building simulation programs use imperative programming, such as procedural code, that defines the sequences of computer instructions as opposed to only the logic of the computation as a declarative language would do.[1] This imperative model formulation does not allow the use of the above symbolic manipulations.

The number of cycles it takes to process an instruction can be reduced by using algorithms that take advantage of parallel hardware. Exploiting parallelism in hardware becomes increasingly important because the third term on the right-hand side is not expected to decrease significantly in the future (Asanovic *et al.* 2006). Increasing parallelism is therefore the primary method of reducing the computing time. Taking advantage of this technological development requires changes in the software architecture in order to implement libraries that exploit parallelism, such as the ones described by Vuduc *et al.* (2005). It also requires adding parallelization constructs to higher level code. The fact that the seconds/clock are not decreasing much but instead the hardware is becoming increasingly parallel has far reaching consequences for the software community (see, for example, Asanovic *et al.* (2006)). Taking advantage of these technological changes is easiest if programs are designed such that they separate concerns for model formulation, symbolic processing, data management and numerical solution. This allows using highly efficient libraries for computational kernels (i.e. computational tasks that are common to many applications such as solvers for linear systems of equations) that can be updated with few changes to the application program as the hardware changes and the state of the art in the scientific computing community advances. It is questionable that traditional building simulation programs, which contain hundreds of thousands of lines of code and whose software architecture has not been designed for parallel computation, can take effective advantage of this development.

4.1.2. *Management of complexity*

The flat model representation that can typically be found in today's building simulation programs is not well suited to manage the complexity of large system models. Instead, a modelling language should allow for the management of the complexity of building system models by providing means for composing system models hierarchically (to encapsulate subsystem models), for object-inheritance (to reuse existing basic models and refine their implementation), for object-instantiation (to use and parameterize an object in a simulation model) and for polymorphism (to change the model semantics based on the environment that the model is exposed to). There should also be a capability that allows a model builder to assemble system models as one would connect components in an actual system, with acausal connections that link model ports that carry physical quantities such as mass flow rate, species concentration, pressure and temperature. These requirements are part of the *object-oriented modelling paradigm* that is described by Cellier (1996) and realized in the Modelica language.

4.1.3. *Simulation of dynamic effects*

Time domain simulation of equipment that switches on and off by averaging its performance over a fixed time step can lead to incorrect prediction of the equipment performance. For example, a DX coil that short-cycles at high frequency has a lower latent heat capacity compared to the same coil that cycles at lower frequency because the water film that deposits on the coil can evaporate into the supply air stream when the compressor is switched off (Henderson and Rengarajan 1996). Furthermore, models that time-average the dynamic performance are not applicable for analysing the robustness and dynamic performance of feedback control loops. Time-averaging the dynamics also limits the use of models in operation for model-based controls, fault detection and diagnostics.

4.1.4. *Use of models beyond time domain simulation*

Models can serve more applications than just time domain simulation. For example, control theory for linear time invariant systems provides a rich framework for systems of the form

$$\dot{x}(t) = A\,x(t) + B\,u(t), \tag{2a}$$

$$y(t) = C\,x(t) + D\,u(t), \tag{2b}$$

where A, B, C and D are matrices with constant coefficients, $x(\cdot)$ is the state vector, $u(\cdot)$ is the control input vector and $y(\cdot)$ is the output vector. In Section 4.6.1 we show how such theory can be used in conjunction with a nonlinear Modelica model for a heat exchanger to design a controller.

4.1.5. *Use of models in conjunction with optimization algorithms*

To prove convergence of optimization algorithms to a stationary point, the cost function needs to be once continuously differentiable. There is a large class of optimization algorithms, some of which do not require knowledge of the gradient, that can be used to efficiently solve such problems (Polak 1997, Kolda *et al.* 2003, Polak and Wetter 2006). It is generally accepted in the optimization community that at least for the final iterates, the simulations need to be done with high enough accuracy of the numerical solvers to make the numerical noise in the cost function negligible. This has been shown to be difficult with building simulation programs that have several solvers spread throughout their code, often without giving the user the means to use tight solver tolerance (Wetter and Polak 2004b, Wetter and Wright 2004) or to adaptively adjust them during the simulation to reduce the computation time such as in Wetter and Polak (2004a).

4.1.6. *Generation of a simulation model from a BIM*

We envision using a Building Information Model to generate a model that can be used for energy simulation from a component-based BIM representation of the building, its HVAC system and its control system. To accomplish this, it seems to be most natural to encapsulate models for simulation in the same way as components are delivered to a building. Furthermore, the rules that describe how component models can be connected to each other to form a system model should be as close as possible to the rules that describe how actual components can be connected to each other.

In view of these characteristics, we suggest revisiting the current approach of writing building simulation programs. Instead of writing programs that describe how a building energy system should be *simulated*, i.e. in what sequence equations are evaluated, how variables are propagated from one routine to another and how equations are being solved, one should write models that define the algebraic and dynamic relationships between their interface variables, and compose system models hierarchically. How to generate a computer code for simulation from these systems of equations should be left to software that manipulates the equations symbolically and links them to numerical solvers. Clearly, this poses formidable challenges to symbolic and numerical solvers. It is the subject of the research described here to explore a model formulation that allows robust and efficient numerical simulations, and to identify research and development needs to make equation-based modelling accessible to a larger community than simulation specialists.

4.2. *Implementation*

To implement our modelling library we selected the Modelica language. Modelica is a free open-source language for object-oriented equation-based modelling of systems that are described by differential, difference and algebraic equations. Its broad support in many industrial domains positions it well to become the de-facto standard for modelling of dynamic systems. Thus, we believe it has the potential to become a language that allows exchanging models among users of different engineering domains, including building technologies, that may use different modelling and simulation environments, similar to what was attempted by Sahlin and Sowell (1989) with the Neutral Model Format language.[2] We note that equation-based modelling and simulation was introduced to the building simulation community close to two decades ago (Sowell *et al.* 1986, Sahlin and Sowell 1989, Charlesworth *et al.* 1991, Klein and Alvarado 1992). What is different from earlier efforts is that Modelica is supported by various industrial sectors which broadens the resources available to develop the language and the modelling and simulation environments. In addition, there have been various advances in symbolic and numerical methods as well as in computer science and computer hardware that make the approach more feasible today.

Before discussing the architecture of our library, we present a brief overview of Modelica and refer for an in-depth discussion to Mattsson and Elmqvist (1997), Tiller (2001) and Fritzson (2004). In 1996, a consortium was formed to develop the Modelica language. The goal of the consortium is to combine the benefits of existing modelling languages and to define a new uniform language for model representation by creating a modelling language that allows modelling systems that involve multiple engineering domains such as electrical engineering, thermodynamics, heat transfer, fluid dynamics and controls (Fritzson and Engelson 1998). Modelica is an equation-based, acausal, object-oriented modelling language that is designed for component-oriented modelling of dynamic systems. Models are described by differential equations, algebraic equations and discrete equations. Using standardized interfaces, the mathematical relations of a model between its interface variables are encapsulated, and the model can be represented graphically by an icon. This encapsulation together with the standardized interface variables facilitate model reuse, model exchange and connecting component models to system models using a graphical or textual editor. Since Modelica is a standardized language, it is a promising choice for ensuring that models can indeed be shared and exchanged by a large user community.

A tenet of Modelica is that each component should represent a physical device with physical interface ports. For ports of opaque heat conductors, interface variables are heat flow and temperature, and for an element that transports a fluid, the port variables are pressure, species flow and enthalpy. This encapsulation together with acausal models enables a graphical, input–output free model construction. In a schematic model diagram of a physical system, icons correspond to actual components or subsystems and encapsulate the equations that define the physics of the subsystem. Lines between the icons impose interface equations to conserve flow and to equate state variables, or they may propagate signals in a control system.

Models can be encapsulated hierarchically. This facilitates managing the complexity of large systems, reusing subsystem models and testing of subsystem models before they are assembled into a large system model that may be difficult to debug. To reduce the model development time, the object-oriented model construction in Modelica allows experts of different domains, such as an HVAC engineer and a controls engineer, to model their respective process, and later interface the models to analyse the complete system. This effectively allows concurrent, as opposed to sequential, model building.

Modelica libraries for multi-domain physics (http://www.modelica.org/libraries/) include models for control systems, for thermal systems, for electrical systems and for mechanical systems, as well as for fluid systems and different media (Elmqvist *et al.* 2003, Casella *et al.* 2006). Because the libraries use standardized interfaces, models from different libraries can be used within the same system model. An example of such a standardized interface is the `HeatPort` connector of the Modelica Standard Library which defines a connector with temperature and heat flow. It is implemented by the following lines of Modelica code:

```
 1  partial connector
 2    Modelica.Thermal.HeatTransfer.
 3        Interfaces.HeatPort
 4        ''Thermal port for 1-D heat
 5          transfer'';
 6      SI.Temperature T
 7        ''Port temperature'';
 8      flow SI.HeatFlowRate Q_flow
 9        ''Heat flow rate (positive if
10          flowing into the component)'';
11  end HeatPort;
```

On line 8, the type prefix `flow` declares that all variables connected to `Q_flow` need to sum to zero. For example, if two `HeatPorts` are connected, the relationship $T_1 = T_2$ and the conservation equation $\dot{Q}_{\text{flow},1} + \dot{Q}_{\text{flow},2} = 0$ are generated. This connector can then be instantiated to define the interface in a one-dimensional heat transfer element with no energy storage. In Modelica's thermal library, such a heat transfer element is implemented as

```
 1  partial model Element1D
 2      ''Partial heat transfer element
 3        with two HeatPort connectors
 4        that does not store energy''
 5      SI.HeatFlowRate Q_flow ''Heat flow
 6        rate from port_a → port_b'';
 7      SI.Temperature dT     ''port_a.T-
 8        port_b.T'';
 9    public
10      HeatPort port_a;
11      HeatPort port_b;
12    equation
13      dT = port_a.T - port_b.T;
14      port_a.Q_flow = Q_flow;
15      port_b.Q_flow = -Q_flow;
16  end Element1D;
```

Lines 5 to 8 contain the declaration of the variables $\dot{Q}$ and ΔT that are typically computed in a one-dimensional heat transfer element. Lines 10 and 11 instantiate the `HeatPort` connector to expose to the outside of this model the port temperatures `port_a.T` and `port_b.T`, as well as the port heat flow rates `port_a.Q_flow` and `port_b.Q_flow`. The equations on lines 13 to 15 define the relationships between the variables of the two `HeatPort` connectors and the variables of the partial model `Element1D`. Note that `Element1D` does not declare a relation between the heat flow rate and the temperatures, as this relation is different for conduction, convection or radiation.

Because this relation is not specified, the model is declared `partial` to indicate that it can be extended by other models to refine its implementation, but that it cannot be instantiated as it does not define its semantic. To implement a thermal conductor, the above partial model can be extended as follows:

```
1 model ThermalConductor
2   ''Lumped thermal element
3    transporting heat without storing it''
4  extends Interfaces.Element1D;
5  parameter SI.ThermalConductance G
6     ''Constant thermal conductance'';
7 equation
8     Q_flow = G* dT;
9 end ThermalConductor;
```

This thermal conductor model can then be encapsulated in a graphical icon using drawing elements that are part of the language standard, and hence, can be interpreted by different Modelica modelling environments. By using a different parameter declaration on line 5 and a different equation on line 8, the semantics can be changed to represent other one-dimensional heat transfer elements such as a model for long-wave radiation between two surfaces. Note that the above code is not pseudo-code, but rather a complete implementation of a heat conductor in the Modelica language. (For simplicity, optional graphical annotations that can be interpreted by a graphical model editor and optional model documentation in html format have been omitted.) Here, a model developer only declared the variables and the constraints between heat flow rate and temperatures. Whether the model will solve for the heat flow rate or for a port temperature will be determined by a code generator that analyses the overall simulation model in order to determine a numerically efficient sequence of computations.

Modelica is a language that defines models but it cannot be executed directly. To create executable code, a Modelica simulation environment translates a Modelica model to a programming language such as C and links it to numerical solvers. See, for example, Cellier and Kofman (2006) for a description of algorithms that are typically used in such a process.

The Software Component Model that is embodied in the Modelica language enables a flexible reuse of models. The Software Component Model consists of *components*, a *connection mechanism* and a *component framework* (Fritzson 2004). Components are connected using Modelica's connection mechanism. This can be visualized in connection diagrams. The component framework realizes components and connections and ensures that communication works over the connections. Connectors can contain physical variables such as temperature and heat flow rate, which are typically implemented using acausal variables. They can also contain signals such as a control input which are typically implemented using causal variables, or they can be composite and involve causal and acausal variables. There are several connectors defined in the Modelica Standard Library, and the design is such that all independent variables that are necessary to define the desired effects in a real interface are part of a connector.

The Software Component Model that is embodied in Modelica allows connecting models as one would connect real components with each other. Therefore, if models are encapsulated the same way as components are encapsulated in a Building Information Model, a one-to-one translation between a BIM and an energy simulation model can be done. With many of today's building simulation programs, this is not possible because building simulation programs impose rules for how component models can be connected to form a system model, and these rules are typically more restrictive than how actual components can be connected to form an HVAC system. An example of such a situation is shown in Figure 1. The figure illustrates a cooling and ventilation system in which components are connected in a way that cannot be directly represented in the EnergyPlus whole building simulation program. The left-hand side of the figure shows the schematic view of the actual system in which the room return air flows through an economizer and then into a double-skin façade from which it is exhausted to the outside by the exhaust air fan. The right-hand side of the figure shows the schematic view of the model with the modifications that were needed to simulate it in EnergyPlus. In particular, an additional economizer had to be added, and the exhaust air fan had to be placed in the supply air stream. This was needed as EnergyPlus does not allow

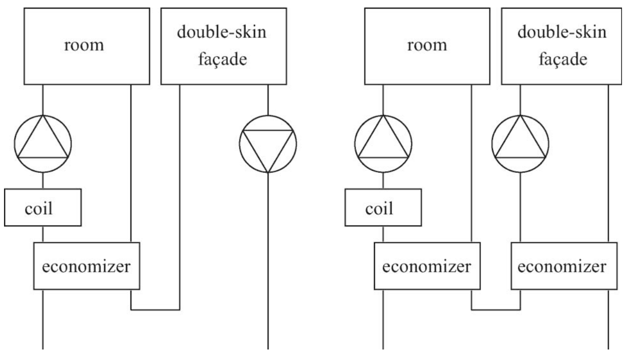


Figure 1. Schematic view of an HVAC system with exhaust air through a double-skin façade (left-hand side) and implementation in the EnergyPlus whole building simulation program (right-hand side).

connecting two thermal zones, one for the room and one for the double-skin façade, with a flow path that contains the economizer. Although an expert modeller may be able to make such modelling decisions, it would be impractical to write a BIM translator that allows such special HVAC system configurations. However, in modular modelling languages such as Modelica, a direct implementation of the actual system is possible as components can be connected in any way as long as the port variables are compatible.

4.3. Base class for library

In Modelica, there are currently two implementations for fluid flow models: the package `Modelica.Thermal.FluidHeatFlow` of the Modelica Standard Library and the package `Modelica_Fluid` (Casella *et al.* 2006). The latter is intended to become part of the Modelica Standard Library.

The package `Modelica.Thermal.FluidHeatFlow` has been developed under the assumption that the mixture concentration remains constant, that the medium does not change phase and that the medium properties are constant. For many building HVAC applications, these assumptions are too restrictive since, for example, change in air humidity or pollutant concentration is often of interest. The package `Modelica_Fluid` is not based on these assumptions. Instead, models in the package `Modelica_Fluid` instantiate a medium model that provides standardized variables and interfaces to medium property functions such as for density or viscosity. Medium models can range in their fidelity from single-phase, single-substance, constant property media to detailed multi-phase, multi-substance media such as air with water in liquid and vapour phase, depending on the water vapour and saturation pressure. Since component models are implemented separately from the medium models, the same component models can be used for different media. Because of this flexibility, our library is based on the package `Modelica_Fluid`. A goal of `Modelica_Fluid` is that every component can be connected in an arbitrary way, and components such as pipes can be reversed without affecting the performance of the simulation model.

`Modelica_Fluid` is still under development, but it already provides a set of component models for one-dimensional thermo-fluid flow in pipe networks. However, the models are not meant to cover all application areas but rather serve as examples for how to implement additional components. In our library, we used some of the same base classes, reused certain models and added additional models that are more specifically targeted towards building HVAC system modelling and simulation. For example, while `Modelica_Fluid` provides spatially discretized heat exchanger models with detailed models for fluid flow friction and convective heat transfer coefficients, we implemented additional heat exchanger models that are more applicable for modelling and simulation of building HVAC systems during early design when little detailed component data are available.

4.4. Buildings library packages

We organized our `Buildings` library into the packages listed below. Additional packages including models that link during the simulation to the Building Controls Virtual Test Bed (Wetter and Haves 2008), and hence to EnergyPlus, will be added later. The library is available from http://simulationresearch.lbl.gov.

`BaseClasses` This package contains base classes that are used by several models of this library such as a basic icon that may be extended by a model to provide a uniform graphical model representation. Note that other packages may have their own sub-package `BaseClasses` that may provide basic models that are used within the corresponding package only. For example, the package `Fluids` has a package `Fluids. BaseClasses` that provides a model for a flow resistance that is used by a fixed flow resistance model and by a valve model.

`Controls` This package contains models of continuous time and discrete time controllers.

`Fluids` This package contains models for fluid flow components such as fixed flow resistances, two- and three-way valves with various opening characteristics, air dampers and fans. There are also models for thermal energy storage tanks, furnaces and measurement sensors.

`Fluids.HeatExchangers` Models in this package include various heat exchanger models including steady-state heat exchangers with fixed effectiveness, dynamic heat exchangers that are discretized along the two flow paths and a simple heater or cooler whose heat transfer rate is proportional to a control input. There are also models for cooling towers.

`Fluids. MassExchangers` This package contains a model for a humidifier and a heat and moisture exchanger.

`HeatTransfer` This package contains models for heat transfer elements such as a finite volume method for heat conduction in solids.

`Media` This package contains media models that can be used in addition to the models provided by `Modelica.Media`. Our models are in general less detailed to reduce simulation time. For example,

the package contains a model for moist air with constant specific heat capacity of air and water vapour, and models for dry and moist air with a simplified implementation of the gas law. There is also a model for water with constant density.
`Utilities` This package provides utility classes that are used by models in several packages. For example, there are psychrometric models. There are also models that facilitate writing results to output files to augment the output capabilities provided by the simulation environment and by the Modelica Standard Library. In addition, this package contains functions that approximate some non-differentiable functions by approximations that are differentiable everywhere and whose derivatives are continuous. Such functions may be used by a model developer to increase numerical efficiency.

4.5. *Technical difficulties*

The formulation of dynamical systems using equation-based languages typically leads to large sparse systems of hybrid differential algebraic equations.[3] A direct solution of these systems is not practical. Instead, the following steps are typically involved to create a computationally efficient simulation program:

(1) The user constructs the model equations using a graphical editor, a textual editor, or a model generator.
(2) State variables are selected and the equations are symbolically manipulated to reduce high index differential algebraic equation systems.
(3) Block lower triangulation and tearing are used to reduce the dimensionality of the linear and nonlinear system of equations.
(4) Program code is generated, compiled and linked to libraries that contain numerical solvers.
(5) The hybrid differential algebraic equation systems are solved to find consistent initial conditions.
(6) The equations are integrated over time.

Steps (2) to (6) can be done automatically without user intervention for simple systems, but they may require user intervention for more complex building energy system models. In step (1), a model builder makes the decision of what physical phenomena are to be included in the model. Here, the level of modelling detail is selected based on the process that should be investigated and the availability, or uncertainty, of the process input data. To guide an algorithm in step (2), a model builder can give hints to a symbolic processor to select state variables and/or to tear equation systems. For example, for a thermodynamic state, either temperature or enthalpy may be used as a state variable, but selecting temperature may lead to non-linear equations that need to be solved numerically, while enthalpy may be expressed as an explicit function of temperature. Such hints may be embedded in a model library. On the basis of our experience, for building energy and control systems, step (5) can be challenging. Here, a user can give guess values to aid the numerical solver in finding consistent initial conditions. Finding proper settings can in some cases require trial and error even for experienced users.

The need for providing good initial guesses currently presents a risk for making equation-based, component-oriented modelling available to a large user base that is not trained in these skills. It also presents a risk to use this technology in conjunction with code generators that may, for example, process a BIM to generate a simulation model for performance assessment, or to use such programs during the operation of the building when they need to run unattended. Possible risk mitigation includes advances in symbolic and numerical methods, embedding good guess values in model libraries, and further research in how models should be formulated to ensure fast and robust numerical simulation. With respect to robust numerical simulation, the current redesign of the `Modelica_Fluid` library shows promising initial results. Since version 1.0 of `Modelica_Fluid`, the library has a new implementation for handling flow reversal in flow networks. The new implementation leads in flow networks to residual equations that are continuous and often differentiable. In the past, these equations were often discontinuous in the iteration variables which frequently caused numerical problems. While further experiments on large and realistic benchmarks are still pending, the initial experiments show significant improvements over the previous implementation in terms of their analytical properties (smoothness of the nonlinear system of equations), and in terms of their numerical performance in computing consistent initial conditions and integrating the equations in time.

We note that solvers in traditional building simulation programs often fail to converge too. A common practice in such cases is to freeze the iteration variable, accept the non-convergent solution and proceed to the next time step. However, as pointed out above, this can cause large numerical noise that can lead to incorrect results when analysis techniques are used that require smoothness of the simulation result with respect to input data, such as a sensitivity analysis for identifying important parameters or non-linear programming for optimization.

Advances are also needed to translate run-time exceptions into a language that can be understood by the model user who may not have a background in numerical methods. Furthermore, diagnostics methods, or expert systems, may be developed that guide a user in finding a better model configuration.

In summary, there are quite a few interesting open research problems in the field of equation-based modelling for building systems. We believe that the benefits of further progress in this area would be substantial as equation-based modelling enables many new applications, it facilitates integration of models from different engineering domains, it allows broadening the development effort for modelling and simulation environments across different disciplines, and it allows modelling at a higher level of abstraction that is closer to how a human typically describes an engineering problem.

4.6. *Applications*

We will now present examples in which we determined control parameters using frequency domain analysis and optimization. The models were constructed using component models of the above described Modelica `Buildings` library, version 4.0.0, with `Modelica_Fluid 1.0 Beta 2` and the Modelica Standard Library 2.2.1. To build, simulate and linearize the system models, we used the Linux version of the Modelica modeling and simulation environment Dymola 6.0b (Brück *et al.* 2002). For the frequency domain analysis, we used MATLAB® 2008a with the Control System Toolbox™ (Mathworks 2008). The optimization was done using GenOpt® 2.1.0 (Wetter 2004).

4.6.1. *Controls design using root locus*

The root locus technique is a commonly used controls design method for linear time invariant (LTI) systems. The root locus shows the location of the poles and zeros of the characteristic equation $1 + k\ G(s) = 0$ for a varying control gain $k \in [0,\infty)$, where $G(s)$ is the open loop transfer function. We will now show the use of this technique to design a controller for a heating coil.

Figure 2 shows the Modelica model of the open-loop system. The circles are boundary conditions with constant pressure and temperature. Their parameters are such that the water flow direction is from the model `sou_1` to `sin_1` and the air flow direction is from `sou_2` to `sin_2`. The control input u is equal to the lift of a valve whose relationship between valve lift and volume flow rate is linear (at a constant pressure difference). The control objective was to track a set point for the heat exchanger air outlet temperature. The dynamic response of the outlet temperature sensor was modelled using a linear first order differential equation. The heat exchanger was a finite volume

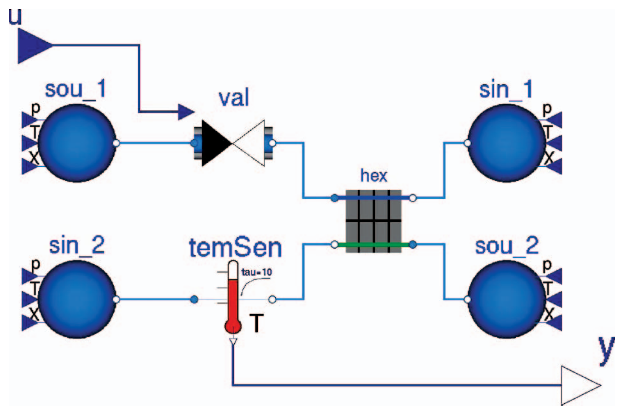


Figure 2. Open loop model for the heating coil.

model with heat capacities for the water in the tubes, for the tube metal wall and for the air. Each pipe was discretized along its water flow path, and the air was discretized along its flow path with an element for each intersection of the air flow path with a pipe. The convective heat transfer coefficient on the water side was a function of the water mass flow rate, whereas the air-side convective heat transfer coefficient was held constant as we did not vary the air flow rate in this example. The valve was sized such that its authority was 0.5. For the closed loop model, we used a proportional controller. The closed loop model defined a differential algebraic equation system with 2593 variables. It was composed of 405 component models (that were primarily used to define the heat exchanger model). There were a total of 36 state variables for the coil model and one state variable for the dynamics of the temperature sensor at the coil air outlet. The biggest coupled nonlinear system of equations was an 8×8 system, which was reduced to a set of scalar equations by the symbolic processor which also found analytic expressions for all elements of the Jacobian matrices.

The closed loop system exhibited oscillatory behaviour for large values of the proportional gain K_p and large steady-state errors for small values. Thus, our objective was to find the value of K_p that gives small steady-state error and non-oscillatory behaviour.

As the plant was nonlinear, we followed the following design procedure:

(1) Bring the open loop system to different steady-state conditions, and linearize the open loop response around these steady-state conditions. This yields, for each linearization, an LTI model of the form $\dot{x} = Ax + Bu$, $y = Cx + Du$, with 37 state variables.
(2) For each LTI model, compute a reduced order model, $\dot{\tilde{x}} = \tilde{A}\,\tilde{x} + \tilde{B}\,u$, $y = \tilde{C}\,\tilde{x} + \tilde{D}\,u$ that has similar response to the original LTI model but

is better suited for controls design than the high order model.

(3) For each reduced order model, design a controller (that is applicable locally because the model is linearized) and then design a control law that is applicable for the whole range of operating conditions.
(4) Test the controller on the original nonlinear model. If the control performance is unsatisfactory, repeat the previous step.

For step (1), we used the Dymola program to bring the plant to steady-state conditions for a small and large valve opening. In particular, we set $u = 0.05$ and $u = 0.95$, respectively, simulated the open loop plant until it reached steady state, and then called the linearization command of Dymola that extracts an LTI model around the current operating point. Thus, rather than implementing a linearized model that would be valid only locally, we implemented a nonlinear model, let it reach two different operating points that were of interest, and then called a linearization command of the simulation environment. The linearization command conducted an input perturbation and reinitialization of state variables as needed during the perturbation to compute a linear approximation to the original model.

For step (2), we imported the LTI model into MATLAB and computed the Hankel singular values$\{\sigma_i\}_{i=1}^{37}$. The first few Hankel singular values were $\sigma_i = \{29.29,\ 2.2997,\ 0.3334,\ 0.2778, \ldots\}$ for $u = 0.05$ and $\sigma_i = \{14.48,\ 1.687,\ 0.5090,\ 0.0217, \ldots\}$ for $u = 0.95$. As the fourth values contributed little to the response, we computed 3rd order reduced order models for both values of u. Figure 3 compares the Bode plots and step responses for the full order and the reduced order models. The figures show that the reduced order models were a good fit.

Next, for step (3), we used the root locus technique to select control gains for each reduced order model. Figure 4 shows the root locus for both reduced order models. From the root locus, we obtained the poles and their damping ratios ζ for different proportional gains K_p as listed in Table 1. The data show that for $K_p \geq 2$ there is little damping at low valve lift.

For step (4), we simulated the original nonlinear system in Dymola for $K_p = 1$ and $K_p = 2$. The steady-state error was 0.06K for $K_p = 1$. For $K_p = 2$, we observed oscillatory behaviour at low valve lift. Thus, we selected a linear gain schedule such that $K_p(u =$

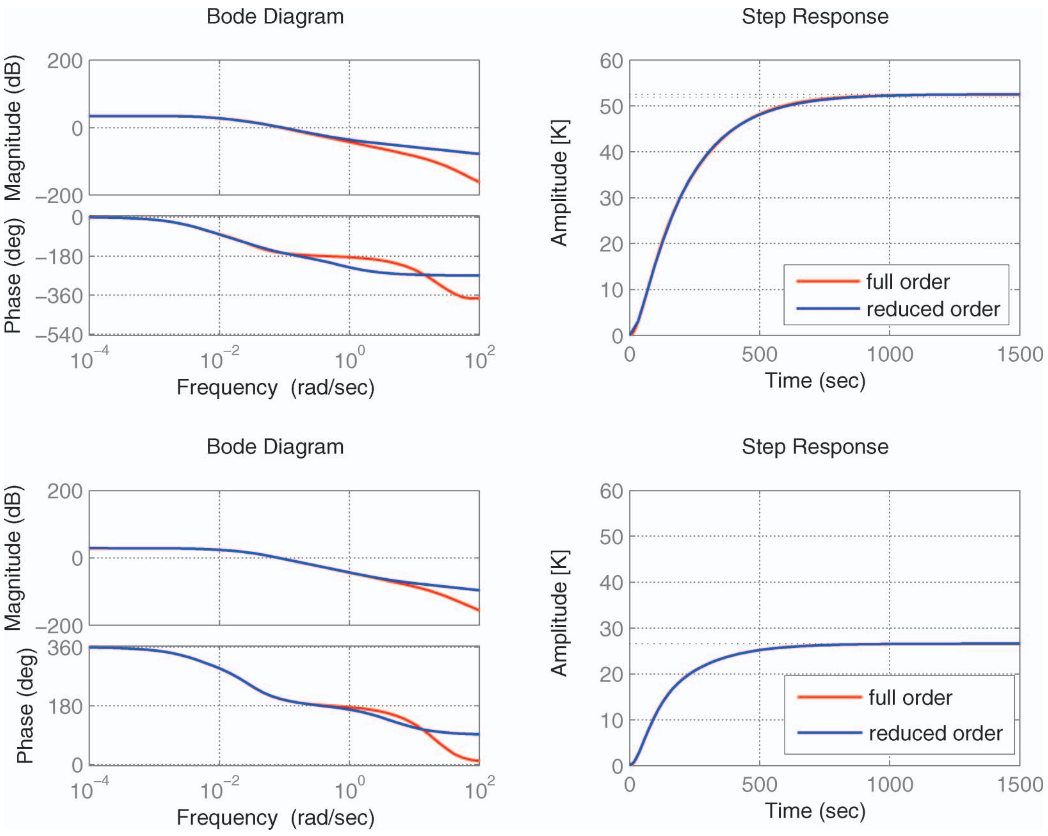


Figure 3. Bode plots and step responses for full order models (with 37 states) and reduced order models (with 3 states). The top row is for $u = 0.05$ and the bottom row for $u = 0.95$.

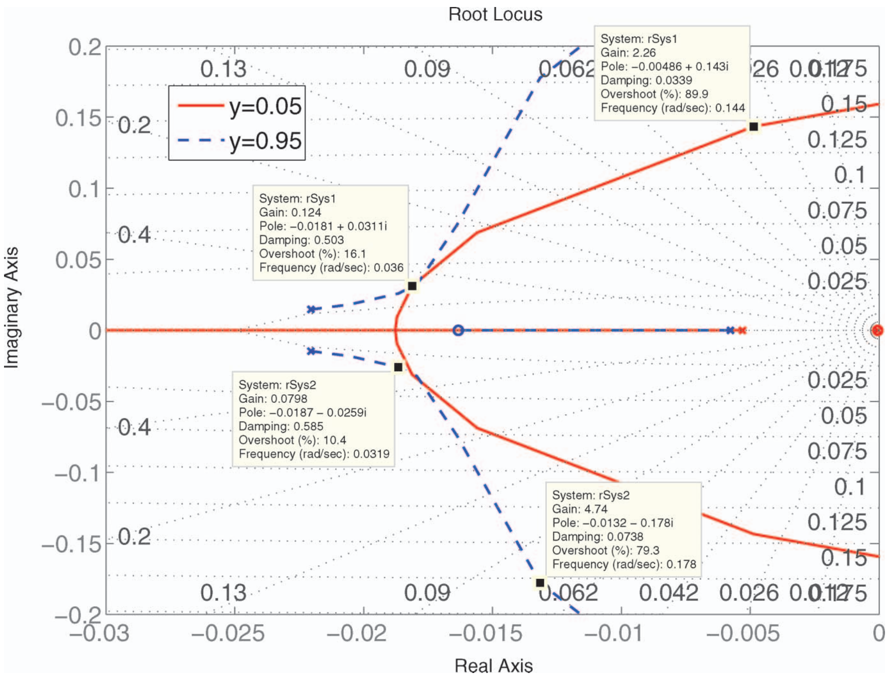


Figure 4. Root locus plot for the reduced order models. The red solid locus corresponds to the system linearized at $u = 0.05$ and the blue dashed locus is for $u = 0.95$.

Table 1. Damping ratio ζ for both reduced order models for different proportional gains K_p. The row ζ ($u = 0.05$) is for the reduced order model that approximates the plant for $u = 0.05$.

K_p	0.13	0.5	1	2	5
$\zeta(u = 0.05)$	0.5	0.23	0.096	0.048	unstable
$\zeta(u = 0.95)$	0.5	0.28	0.19	0.13	0.07

$0.05) = 1$ and $K_p(u = 0.95) = 2$. This linear gain schedule resulted in the step response shown in Figure 5, with a steady-state error below 0.4K. To reduce the valve oscillation after the step signal at $t = 30$ min would require a reduction in K_p and hence would lead to a larger steady-state error. Thus, if the transient response in Figure 5 is not acceptable, then the proportional controller would have to be replaced, for example, by a proportional-integral controller.

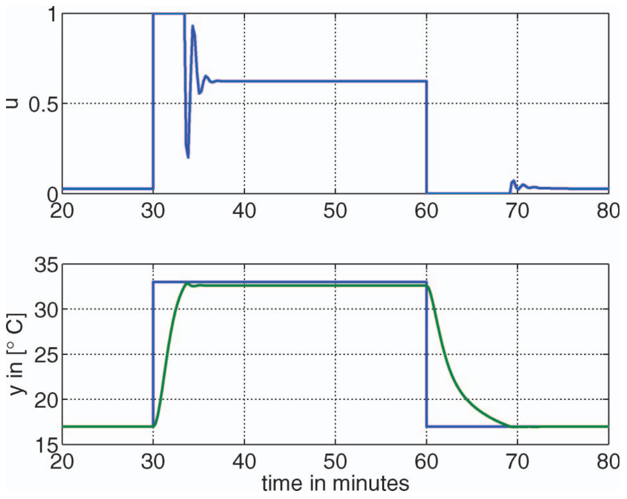


Figure 5. Closed loop step response with gain scheduling such that $K_p(u = 0.05) = 1$ and $K_P(u = 0.95) = 2$. The top figure is the actuator input, and the bottom figure is the temperature measured at the coil outlet.

4.6.2. *Precommissioning of a fan static pressure reset controller for a Variable Air Volume flow system*

For Variable Air Volume (VAV) flow systems with direct digital control to the zone level, the California Title 24 Energy Code requires supply pressure reset by zone demand (CEC 2005). A control logic that is used by major control vendors is the Trim and Response logic (Taylor 2007). This control logic may be implemented as follows. If the supply fan is on, a central controller samples the VAV damper positions every 2 min. If no damper is close to fully open, the pressure set point is decreased by 10 Pascals; else it is increased by 15 Pascals. When the fan switches off, the set point is set to 125 Pascals.

Since the default parameters for the pressure adjustment will work well only for few systems, a trial and error tuning is almost always required during commissioning (Taylor 2007). During commissioning, the engineer adjusts pressure increments and decrements until the system is stable.

We will now perform this procedure using a Modelica model that we linked to the GenOpt optimization program to find appropriate settings that may be used as a starting point during a commissioning process. For this example, we created a Modelica implementation of the VAV system model described in ASHRAE 825-RP (Haves *et al.* 1996). Unlike the model used by Haves *et al.* (1996), our model also includes equations for the CO_2 concentration, in addition to equations for enthalpy (or temperature), pressure and mass flow rate. We also added models of completely mixed air volumes to model the CO_2 storage in the room air and in the return air plenum. Figure 6 shows the system model in Modelica. On top are the controllers for the static pressure reset and the fan frequency drives. The blue circles on the left are ambient conditions, which are connected to an outside air mixing box. There are also flow resistances for the ducts and the supply and return fans. On the right is a system model that encapsulates the supply ducts, the return ducts and the six rooms.

Figure 7 shows the implementation of the room models. Each room has an individual VAV damper with nonlinear characteristics between opening angle and flow rate. The model with label `vav` contains the VAV damper and a flow resistance that has been parameterized to account for the duct and diffuser resistance. This model is connected to an instantaneously mixed room volume which is labeled `vol`. Counterclockwise, this room volume model is connected to a model for the room leakage to the exterior (with label `lea`), to a port that allows connecting different rooms with each other (using a flow resistance to separate adjacent rooms), to a CO_2 emission model (its input is the number of people), and to another mixed air volume that models the CO_2 concentration in the exhaust air plenum. Next, there is another port that is used to connect the room volume to another room to its right and to a CO_2 sensor. The CO_2 sensor is connected to a model that converts mass fraction to volume fraction. Its output is connected to a proportional controller with saturation. The controller output is connected to a model for the actuator motor, which has a finite travel speed and a hysteresis that causes it to adjust the damper position only if the position error exceeds a threshold. This configuration is used for each of the six rooms.

The six rooms are then connected with each other using a pressure drop element to model interzonal air exchange, and they are connected to an air distribution system for the supply and return duct. This subsystem model is then encapsulated into the icon with label `roo` in Figure 6 and connected to the plant model to form the overall system model.

In the system model, all air mass flow rates are computed based on the flow friction and the fan curve that relates fan volume flow rate with fan pressure raise for the given number of revolutions which is determined by a model for the fan frequency controller. The total system model contained 730 simulation components that led to a differential algebraic equation system with 4420 scalar equations and 40 state variables. Some models triggered time events (e.g. for discrete time controls) or state events (e.g. for the hysteresis of the damper position adjustment). The computation time to

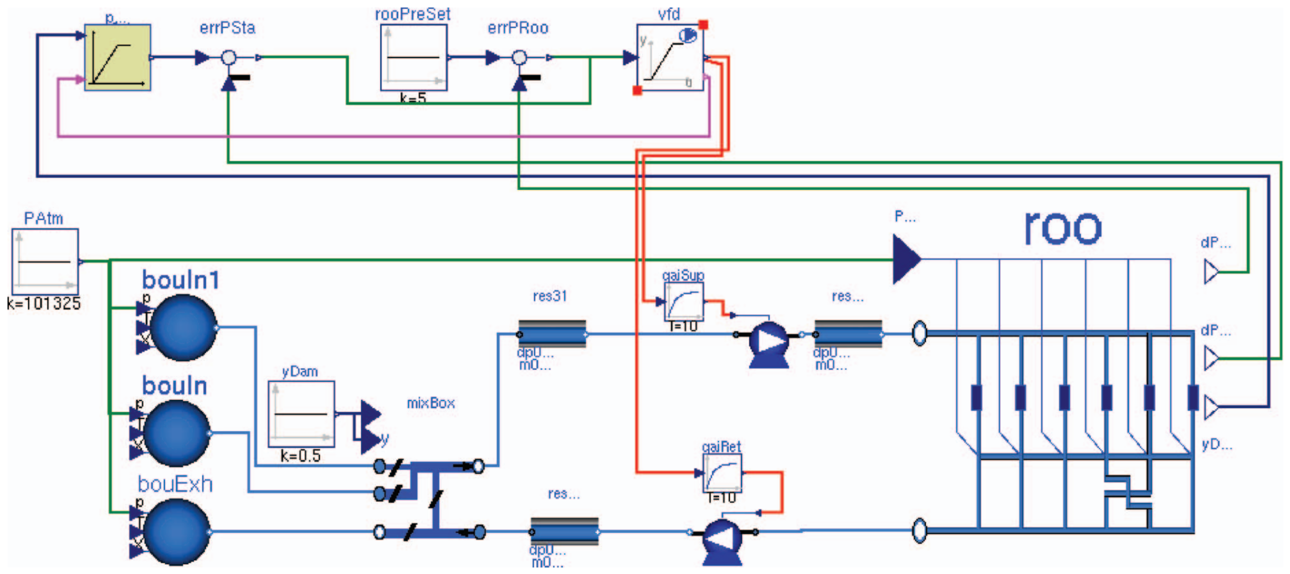


Figure 6. Implementation of VAV system model in Modelica.

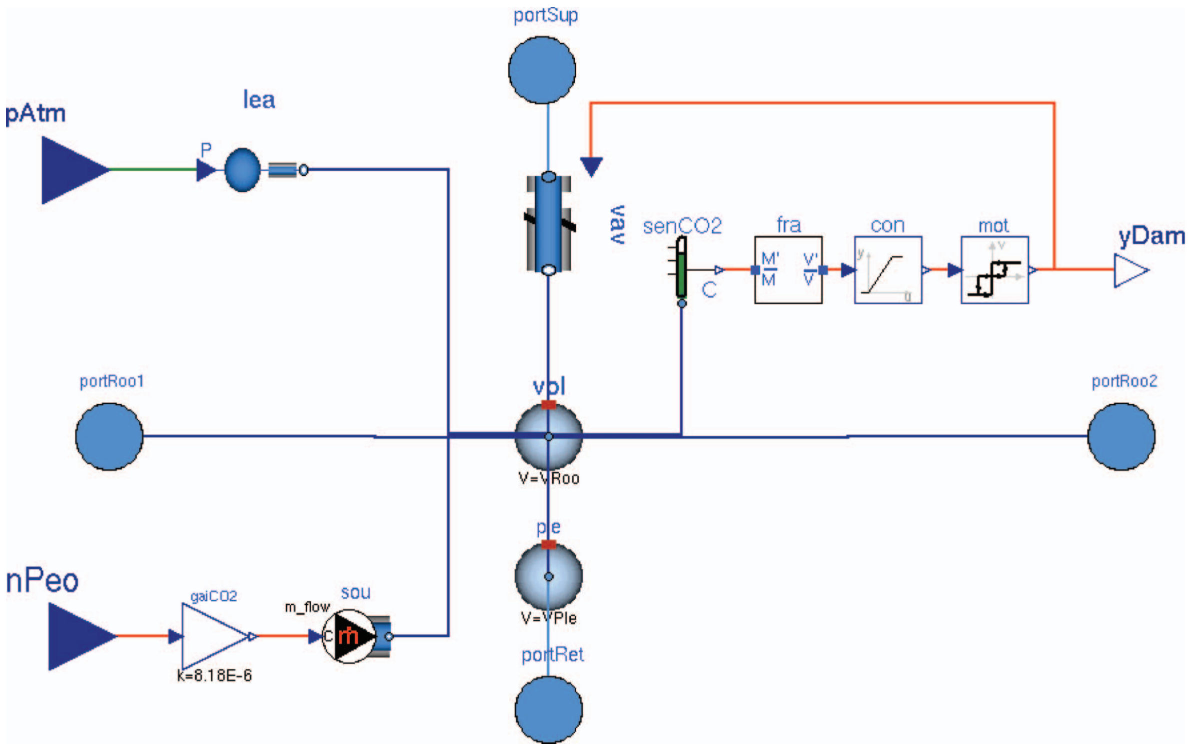
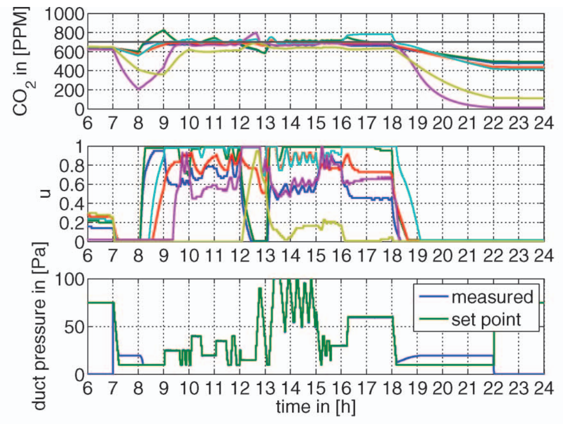


Figure 7. Implementation of the room models in Modelica.

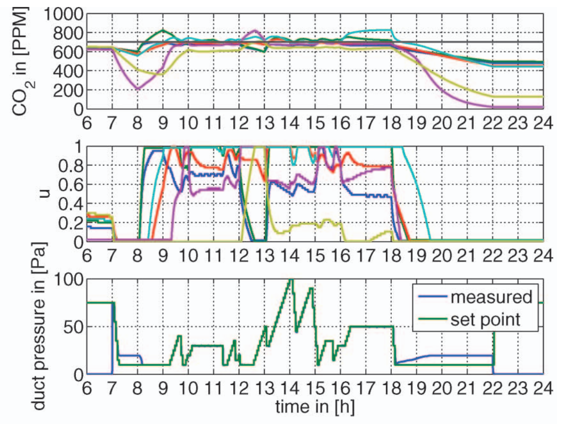
simulate one day was about 10 min. One of the main reasons for the large computation time was the discrete time control that led to time events, and the motor model of the VAV damper that triggered state events due to its hysteresis model. In total, there were 1440 time events and 5172 state events. The time integration step size, which is adaptive, was never enlarged to more than 30 s, most likely due to the events. Thus, while this system model is applicable for controls analysis, one would hardly use such a level of modelling detail for an annual simulation. Rather, one may replace the discrete time control with continuous time control to reduce the time event, neglect the hysteresis of the motor to eliminate state events, and introduce a transport delay in the central plant's supply and return duct to break a 50 dimensional nonlinear equation system that solves for pressure and mass flow rates into three smaller equation systems with dimensions 9, 15 and 21. When we implemented these changes (in a more recent model that was based on `Modelica_Fluid` version 1.0, release candidate 1), the computation time was reduced by a factor of 50, and the numerical solver was able to enlarge the integration step size to a maximum of 1100 s.

To simplify the example, we assumed that thermal conditioning and ventilation were provided by separate systems. Thus, the one control objective was to maintain a CO_2 concentration in each room of 700 PPM above the outside air concentration by regulating the room VAV dampers, and to minimize the fan static pressure using the Trim and Response logic. We assumed a fixed outside air damper position of $u = 0.5$, an occupant density of up to 0.15 person per m^2 floor area, which varied over the day for each zone individually. A CO_2 emission rate per person of $8.18 \cdot 10^{-6}$ kg/s (= 15dm^3/h CO_2 emission per person) was used (Lochau 1989). The VAV boxes are pressure dependent, i.e., their flow rate changes if the difference between inlet and room static pressure changes. Each room has a continuous-time proportional controller that tracks the room CO_2 concentration. Figure 8a shows the time response of the system with base case controller settings as listed in Table 2. The figure shows that during the occupied hours, there was a frequent change in duct static pressure (bottom figure) which caused an oscillatory behaviour of the VAV dampers (middle figure).

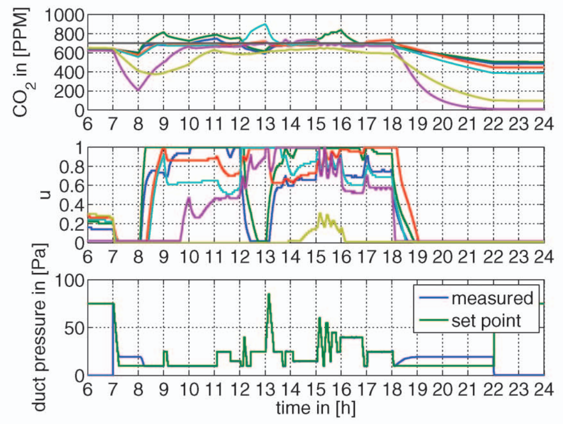
Next, to avoid the oscillatory VAV actuator movement, we adjusted the sampling time t_s, the pressure increment δp_i and decrement δp_d of the Trim and Response algorithm. We note that because of the nonlinearity of fan power consumption with respect to duct static pressure set point, on a temporal average, an oscillatory duct static pressure setpoint causes more flow friction compared to a stable control, and hence leads to a larger fan energy use. Thus, instead of a manual trial and error procedure to find values for t_s, δp_i and δp_e that lead to a stable control, we used optimization to find the value for $x \triangleq (t_s, \delta p_i, \delta p_e)$ that minimizes the fan energy use, subject to a constraint



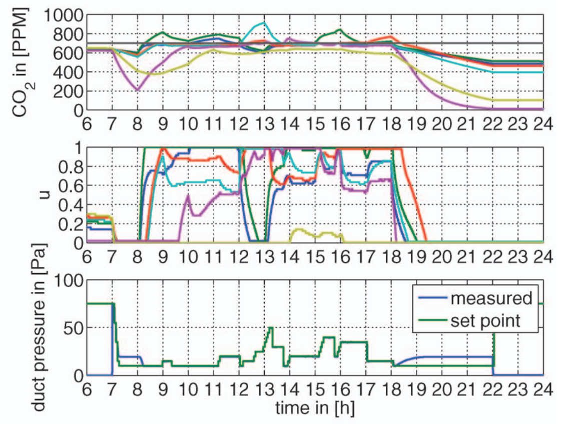
(a) Base case parameter settings x_0, with CO$_2$ emission profiles as used in optimization.



(b) Optimized parameter settings x^*, with CO$_2$ emission profiles as used in optimization.



(c) Base parameter settings x_0, with CO$_2$ emission profiles different from what was used when doing the optimization.



(d) Optimized parameter settings x^*, with CO$_2$ emission profiles different from what was used when doing the optimization.

Figure 8. CO$_2$ concentrations above outside air concentration in the six rooms (with the set point shown in black), VAV damper positions and duct static pressure for the base case and for the control settings obtained using optimization.

on the room CO$_2$ concentration. In particular, we solved

$$\min_{x \in \mathbf{X}} \{ f(x) \mid g(x) = 0 \}, \tag{3}$$

where

$$f(x) = \frac{1}{E_0} \int_0^T P_f(x,t)\, dt \tag{4}$$

is the normalized fan energy use, **X** is the feasible set that imposes upper and lower bounds for the independent variables, $T = 1$ day is the simulation period and $P_f(\cdot,\cdot)$ is the sum of the supply and return fan power consumption. The term $E_0 = \int_0^T P_f(x_0,t)dt$ is the fan energy consumption at the initial iterate x_0. It is used to normalize the cost function which will facilitate the scaling of the constraint function $g(\cdot)$. The constraint function g: $\mathbb{R}^3 \rightarrow \mathbb{R}$ ensures that the CO$_2$ set point in the rooms is not exceeded, except for a possible user-specified number of extreme zones in which the CO$_2$ may not be tracked. Because we configured the Trim and Response logic so that at least three rooms require an increase in static pressure before the fan static pressure set point is increased, we defined the constraint function as

$$g(x) = \frac{1}{T} \int_0^T (\max\{0, \, (y_j(x,t)/\hat{x}_s) - 1/(2\,K_p) - 1 \mid j \in \mathbf{J}(x,t)\})^2\, dt, \tag{5}$$

where $\mathbf{J}(\cdot\,,\,\cdot)$ is the index set of all VAV dampers except the two with the biggest set point violation, $y_j(\cdot\,,\,\cdot)$ is the measured room CO$_2$ concentration, $\hat{x}_s = 700$ PPM is the CO$_2$ set point (above the outside air concentration) and $K_p = 10$ is the proportional gain of the P controller for the room air dampers. The term $1/(2K_p)$ is used to account for the steady-state error associated with proportional controllers.

To compute a solution for (3), we solved the sequence of optimization problems

$$\mathbf{P}_k \qquad \min_{x \in \mathbf{X}} \theta_k(x), \tag{6}$$

where $\theta_k(\cdot)$, for $k \in \mathbb{N}$, is the optimality function, defined as $\theta_k(x) \triangleq f(x) + k^4 g(x)$ for $k \in \mathbb{N}^4$. We solved (6) for $k \in \{1,2,3,4\}$, using the GenOpt optimization program. Parametric studies showed that $f(\cdot)$ is non-convex and may have local minima. We also note that $f(\cdot)$ is non-differentiable because of the state-events and time-events. Therefore, we used GenOpt's Particle Swarm Optimization algorithm PSOCCMesh, which is a global optimization algorithm for non-differentiable cost functions. We will denote by x^* the best iterate found when solving P_k and by x_0 the initial iterate.

In addition to doing the optimization, we also tested how robust the optimal parameter settings are with respect to changes in the CO_2 emission profiles. For this robustness test, for each room we changed the time profiles and their magnitude. For the new set of profiles, we conducted two simulations, one with the initial iterate x_0 and one with x^* which was obtained when solving the optimization problem (6) with the *original* set of CO_2 emission profiles. Thus, x^* need not be optimal for the new set of CO_2 emission profiles, but our objective is to see if x^* also leads to good performance under this new set of CO_2 emission profiles. The performance indices were $f(\cdot)$, as defined in (4), and the normalized distance $s(\cdot)$ travelled by the actuators of all VAV dampers during the simulation period.

Table 2 lists the initial values and the final values of the optimization, together with the results of the optimization and the robustness test. For both CO_2 emission profiles, the optimized parameter settings reduced the fan energy consumption by 3% to 4% and the distance $s(\cdot)$ by about 14%. For the modified set of CO_2 emission profiles, the fan energy was significantly smaller because less CO_2 was emitted. Figure 8 also shows that the optimized parameter settings x^* led to fewer oscillations of the VAV dampers because of a less aggressive re-adjustment of the static pressure set point. The CO_2 concentrations are, however, still maintained within the acceptable limit, i.e., $g(x) = 0$ for all four simulations. There is a steady-state control error between 18:00 and 22:00, as shown in Figure 8, because during this time period, the fan is still on (as it is operated on a timer) while all VAV boxes are closed.

Table 2. Base case and results of the control parameter optimization.

	original CO_2 profile		modified CO_2 profile	
	base case, x_0	optimum, x^*	base case, x_0	optimum, x^*
sampling time t_s in [s]	120	220	120	220
pressure increment δ_i in [Pa]	15	5	15	5
pressure decrement δ_d in [Pa]	-10	-20	-10	-20
fan energy use $f(x)$	1	0.966	0.642	0.614
constraint violation $g(x)$	0	0	0	0
distance $s(x)$	1	0.853	0.816	0.702

5. Conclusions

We showed that equation-based object-oriented modelling allows analysing problems that are beyond the capabilities of traditional building simulation programs. Compared to earlier efforts to establish equation-based object-oriented modelling in the building simulation community, we believe that the development of Modelica, which is supported across many engineering domains, provides an opportunity to revive the effort of establishing more modular and flexible modelling and simulation techniques for building energy and control systems. As Modelica is an open-source language for which several commercial and open-source modelling and simulation environments are both existing and emerging, it provides an open environment for collaboration and model exchange.

Although there are many challenges to be overcome, we believe that equation-based object-oriented modelling allows the field of building simulation to progress towards the next generation of tools. Such a progression is similar to the developments in computer science in which the level of abstraction at which programs are formulated increased from machine language to symbolic assembler, to macro processors, to high-level procedural languages (such as FORTRAN) and further to object-oriented languages (Shaw and Garlan 1996). It would be rather surprising if the field of building simulation were to stagnate at procedural languages that frequently led to large monolithic simulation programs, which are hard to maintain and for which adding new models and analysis features is a significant time investment. A more flexible modelling environment also has the potential to better support design processes that reduce cost and development time, such as model-based system-level design, in which systems are expressed at an increasingly higher level of abstraction to allow a designer to focus on the system architecture rather than its details of implementation (Sangiovanni-Vincentelli 2007).

However, further research and development is needed to make available equation-based object-oriented modelling and simulation to a larger audience for whom debugging non-convergent models is not an option. Such research and development is part of the future work on the Modelica-based library for building energy and control systems described here.

To further disseminate equation-based object-oriented modelling and simulation to the building simulation community, the question, however, is not how to get existing users to adapt to a new technology, but rather how to better integrate modelling into rapid prototyping for inventing new building systems, and into the design and operation of buildings. New technologies are seldom introduced by forcing a change onto existing users, but rather through the enabling of processes that add value for the user and that were not possible before.

Nomenclature

Conventions

(1) Elements of a set or a sequence are denoted by subscripts.
(2) $f(\cdot)$ denotes a function where $(\cdot)$ stands for the undesignated variables. $f(x)$ denotes the value of $f(\cdot)$ for the argument x. $f: A \to B$ indicates that the domain of $f(\cdot)$ is in the space A, and that the image of $f(\cdot)$ is in the space B.
(3) We say that a function $f: \mathbb{R}^n \to \mathbb{R}$ is once continuously differentiable if $f(\cdot)$ is defined on $\mathbb{R}^n$, and if $f(\cdot)$ has a continuous derivative on $\mathbb{R}^n$.

Symbols

$f(\cdot)$	cost function
$g(\cdot)$	constraint function
$\theta_k(\cdot)$	optimality function, for $k \in \mathbb{N}$
K_p	proportional gain
$G(s)$	open loop transfer function
t	time
u	control signal
x	independent parameter
x_k	iterate of the optimization algorithm, for $k \in \mathbb{N}$
x^*	best iterate found by the optimization algorithm
y	measurement signal
σ	Hankel singular values of the linearized system
ζ	damping ratio
$a \in \mathbf{A}$	a is an element of $\mathbf{A}$
$\mathbb{N}$	$\{0,1,2,\ldots\}$
$\mathbb{R}$	set of real numbers
$\triangleq$	equal by definition

Acknowledgements

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the US Department of Energy, under Contract No. DE-AC02-05CH11231. The author would like to thank Scott A. Bortoff at the United Technologies Research Center for his input to the controls design example and Philip Haves, Brian A. Coffey and Walter F. Buhl, all at the Lawrence Berkeley National Laboratory, as well as the anonymous reviewers, for their valuable feedback to the manuscript.

Notes

1. For example, the statement `Q1:=-Q2-Q3;` is imperative and describes that Q1 is to be computed by assigning the sum of `-Q2` and `-Q3`. A declarative statement may have the form `0 = Q1+Q2+Q3;` which only describes *how* the three variables are related, but it does not specify what computer instructions need to be done to compute one from the other two.
2. Within the ASHRAE Technical Committee 4.7, the prospect of Modelica was already mentioned in 1998 when Sowell reported on this language. The committee reported that Modelica could eventually subsume NMF but it was decided to press for NMF and monitor the Modelica progress (Spitler 1998).
3. By hybrid differential algebraic systems of equations, we mean differential and difference equations that are coupled to continuous and discrete algebraic equations. An example is a chiller control that may be expressed as a state machine that is linked to a discrete time controller for a mechanically cooled building, which may be described by differential and algebraic equations.
4. The term k^4 adds a penalty to $f(\cdot)$, if constraints are violated, that is increasing in k, for $k \in \mathbb{N}$.

References

Asanovic, K., *et al.*, 2006. *The landscape of parallel computing research: a view from Berkeley*. Technical report UCB/EECS-2006-183. Berkeley: EECS Department, University of California.

ASHRAE, 2005. *Sequences of operation for common HVAC/Systems*. Atlanta, GA. ISBN 1-931862-98-2.

Banaszuk, A., Mehta, P.G., and Hagen, G., 2007. The role of control in design: from fixing problems to the design of dynamics. *Control Engineering Practice*, 15 (10), 1292–1305.

Bazjanac, V. and Maile, T., 2004. IFC HVAC interface to EnergyPlus – a case of expanded interoperability for energy simulation. *In*: *Proceedings of SimBuild 2004*, Aug, Boulder, CO.

Brück, D., *et al.*, 2002. Dymola for multi-engineering modeling and simulation. *In*: M. Otter, ed. *Proceedings of the 2nd Modelica conference*, Mar, Oberpfaffenhofen, Germany. 55–1–55–8.

Bunus, P. and Fritzson, P., 2004. Automated static analysis of equation-based components. *Simulation*, 80 (7–8), 321–345.

Casella, F., *et al.*, 2006. The Modelica fluid and media library for modeling of incompressible and compressible thermo-fluid pipe networks. *In*: C. Kral and A. Haumer, eds. *Proceedings of the 5-th International Modelica Conference*, Vol. 2, Sep, Vienna, Austria, 631–640.

CEC, 2005. *Building energy efficiency standards, nonresidential compliance manual*. Sacramento, CA.

Cellier, F.E., 1996. Object-oriented modeling: means for dealing with system complexity. *In*: *Proceedings 15th Benelux Systems and Control Conference*, Mierlo, The Netherlands, 53–64.

Cellier, F.E. and Kofman, E., 2006. *Continuous system simulation*. New York: Springer.

Charlesworth, P., *et al.*, 1991. The energy kernel system. *In*: J.A. Clarke, J.W. Mitchell, and R.C.V. de Perre, eds. *Proceedings of the IBPSA Conference*, Aug, Nice, France.

CIBSE, 2000. *Building control systems, CIBSE Guide H*. London: Butterworth-Heinemann.

Clarke, J.A., 2001. *Energy simulation in building design*. 2nd ed. Oxford, UK: Butterworth-Heinemann.

Crawley, D.B., *et al.*, 2001. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33 (4), 443–457.

Elmqvist, H., Otter, M., and Cellier, F., 1995. Inline integration: a new mixed symbolic/numeric approach for solving differential– algebraic equation systems. *In*: *Keynote Address, Proceedings. ESM'95*, Jun, Prague, CzechRepublic, xxiii–xxxiv.

Elmqvist, H., Tummescheit, H., and Otter, M., 2003. Object-oriented modeling of thermo-fluid systems. *In*: P. Fritzson, ed. *Proceedings of the 3rd Modelica conference*, Nov, Linköping, Sweden, 269–286.

Felgner, F., *et al.*, 2002. Simulation of thermal building behaviour in Modelica. *In*: M. Otter, ed. *Proceedings of the 2nd Modelica conference*, Mar, Oberpfaffenhofen, Germany, 147–154.

Fritzson, P., 2004. *Principles of object-oriented modeling and simulation with Modelica* 2.1. Piscataway, NJ: Wiley.

Fritzson, P. and Engelson, V., 1998. Modelica – a unified object-oriented language for system modeling and simulation. *Lecture Notes in Computer Science*, 1445.

Hairer, E. and Wanner, G., 1996. *Solving ordinary differential equations. II*. 2nd Springer series in computational mathematics. Berlin: Springer-Verlag.

Haves, P., *et al.*, 1996. *A standard simulation testbed for the evaluation of control algorithms & strategies*. Final Report 825-RP, ASHRAE, Atlanta, GA.

Henderson, H. and Rengarajan, K., 1996. A model to predict the latent capacity of air conditioners and heat pumps at part- load conditions with constant fan operation. *ASHRAE Transactions*, 102 (1), 266–274.

Hoh, A., *et al.*, 2005. A combined thermo-hydraulic approach to simulation of active building components applying Modelica. *In*: G. Schmitz, ed. *Proceedings of the 4th Modelica conference*, Mar, Hamburg, Germany.

Klein, S.A. and Alvarado, F.L., 1992. *Engineering equation solver (EES)*.

Kolda, T.G., Lewis, R.M., and Torczon, V., 2003. Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review*, 45 (3), 385–482.

Lochau, R., 1989. *Physiologische Grundlagen*. *In*: W. Leiner, H. Schedwill, G. Seng, and W. Stäbler, eds. *Handbuch der Klimatechnik*, Vol. 1. Karlsruhe, Germany: Verlag C.F. Müller GmbH, 93.

Mathworks, 2008. *Control System Toolbox 8.2*, Natick, MA.

Mattsson, S.E. and Elmqvist, H., 1997. Modelica – an international effort to design the next generation modeling language. *In*: L. Boullart, M. Loccufier, and S.E. Mattsson, eds. *7th IFAC Symposium on Computer Aided Control Systems Design*, Apr, Gent, Belgium.

Merz, R.M., 2002. *Objektorientierte modellierung thermischen Gebäudeverhaltens*. Thesis (PhD). Universität Kaiserslautern.

Nytsch-Geusen, C., *et al.*, 2005. A hygrothermal building model based on the object-oriented modeling language Modelica. *In*: I. Beausoleil-Morrison and M. Bernier, eds. *Proceedings of the Ninth International IBPSA Conference*, Vol. 1, Aug, Montreal, Canada. 867–876.

Polak, E., 1997. *Optimization, algorithms and consistent approximations*. Applied Mathematical Sciences Vol. 124. New York: Springer Verlag.

Polak, E. and Wetter, M., 2006. Precision control for generalized pattern search algorithms with adaptive precision function evaluations. *SIAM Journal on Optimization*, 16 (3), 650–669.

Sahlin, P. and Sowell, E.F., 1989. A neutral format for building simulation models. *In*: *Proceedings of the Second International IBPSA Conference*, Jun, Vancouver, BC, Canada, 147–154.

Sangiovanni-Vincentelli, A., 2007. Quo Vadis, SLD? Reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95 (3), 467–506.

Shaw, M. and Garlan, D., 1996. *Software architecture: perspectives on an emerging discipline*. New Jersey: Prentice Hall.

Sowell, E.F., *et al.*, 1986. *A prototype object-based system for HVAC simulation*. Technical report LBL-22106, Lawrence Berkeley National Laboratory.

Sowell, E.F. and Haves, P., 2001. Efficient solution strategies for building energy system simulation. *Energy and Buildings*, 33 (4), 309–317.

Spitler, J.D., 1998. ASHRAE TC 4.7 Minutes.

Taylor, S.T., 2007. VAV system static pressure setpoint reset. *ASHRAE Journal*, 24–32.

Tiller, M.M., 2001. *Introduction to physical modeling with Modelica*. Norwell, MA: Kluwer Academic Publisher.

Trcka, M., Hensen, J.L.M., and Wijsman, A.J.T.M., 2006. Distributed building performance simulation - a novel approach to overcome legacy code limitations. *ASHRAE HVAC&R*, 12 (3a), 621–640.

Trcka, M., Wetter, M., and Hensen, J., 2007. Comparison of co-simulation approaches for building and HVAC/R Simulation. *In*: J. Yi, Z. Yingxin, Y. Xudong, and L. Xianting, eds. *Proceedings of the 10-th IBPSA Conference*, Beijing.

Vuduc, R., Demmel, J.W., and Yelick, K.A., 2005. OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16, 521–530.

Wetter, M., 2004. *GenOpt, generic optimization program, User Manual, Version 2.0.0*. Technical report LBNL-54199. Berkeley, CA, USA: Lawrence Berkeley National Laboratory.

Wetter, M., 2006a. Multizone airflow model in Modelica. *In*: C. Kral and A. Haumer, eds. *Proceedings of the 5th International Modelica Conference*, Vol. 2, Sep, Vienna, Austria, 431–440.

Wetter, M., 2006b. Multizone building model for thermal building simulation in Modelica. *In*: C. Kral and A. Haumer, eds. *Proceedings of the 5th International Modelica Conference*, Vol. 2, Sep, Vienna, Austria, 517–526.

Wetter, M. and Haves, P., 2008. A modular building controls virtual test bed for the integration of heterogeneous systems. *In*: *Proceedings of SimBuild*, Aug, Berkeley, CA.

Wetter, M., *et al.*, 2008. Using SPARK as a solver for Modelica. *In*: *Proceedings of SimBuild*, Aug, Berkeley, CA.

Wetter, M. and Polak, E., 2004a. Building design optimization using a convergent pattern search algorithm with adaptive precision simulations. *Energy and Buildings*, 37 (6), 603–612.

Wetter, M. and Polak, E., 2004b. A convergent optimization method using pattern search algorithms with adaptive precision simulation. *Building Services Engineering Research and Technology*, 25 (4), 327–338.

Wetter, M. and Wright, J., 2004. A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. *Building and Environment*, 39 (8), 989–999.

Winkelmann, F.C., *et al.*, 1993. *DOE-2 Supplement, Version 2.1E*. Technical report LBL-34947, Lawrence Berkeley National Laboratory, Berkeley, CA, USA.