A Modelica-based Model Library for Building Energy and Control Systems

Michael Wetter, Lawrence Berkeley National Laboratory

July 2009

DISCLAIMER

A MODELICA-BASED MODEL LIBRARY FOR BUILDING ENERGY AND CONTROL SYSTEMS

Michael Wetter
Simulation Research Group, Building Technologies Department
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA

ABSTRACT

This paper describes an open-source library with component models for building energy and control systems that is based on Modelica, an equation-based object-oriented language that is well positioned to become the standard for modeling of dynamic systems in various industrial sectors. The library is currently developed to support computational science and engineering for innovative building energy and control systems. Early applications will include controls design and analysis, rapid prototyping to support innovation of new building systems and the use of models during operation for controls, fault detection and diagnostics.

This paper discusses the motivation for selecting an equation-based object-oriented language. It presents the architecture of the library and explains how base models can be used to rapidly implement new models. To demonstrate the capability of analyzing novel energy and control systems, the paper closes with an example where we compare the dynamic performance of a conventional hydronic heating system with thermostatic radiator valves to an innovative heating system. In the new system, instead of a centralized circulation pump, each of the 18 radiators has a pump whose speed is controlled using a room temperature feedback loop, and the temperature of the boiler is controlled based on the speed of the radiator pump. All flows are computed by solving for the pressure distribution in the piping network, and the controls include continuous and discrete time controls.

INTRODUCTION

To significantly reduce greenhouse gas emissions associated with building operations, development of building simulation programs along two parallel tracks is needed: First, the usability of existing building simulation programs needs to be improved so that they can better support the design of efficient buildings on a large scale. Second, to accelerate the innovation of new HVAC components, systems and control algorithms, a modeling and simulation framework needs to be developed that better meets the functional requirements of such applications. This paper deals with the second development. Typical functional requirements include:

1. Faster implementation of models for equipment, systems and control algorithms, at different levels of abstraction.
2. Means for implementing models by tool users in addition to tool developers.
3. Ability to share models among users.
4. Ability to model continuous time dynamics (for physical processes), discrete time systems (for discrete time controls) and state events (such as for switching controls).
5. Extraction of subsystem models for use in isolation from the total system model (such as for validation, for more refined analysis, for model reduction, or for use in operation).
6. Use of simulation models in conjunction with nonlinear programming algorithms that require the cost function (such as energy use) to have little numerical noise. This is important to efficiently solve optimal control problems that may involve state trajectory constraints and hundreds of independent parameters that define the control function.

From these functional requirements result several characteristic features of the software architecture that such a modeling and simulation environment should have. They include:

1. Object-oriented modeling to facilitate code reuse.
2. Use of an equation-based language to allow more natural modeling.
3. Use of a standardized modeling language.
4. Support for interfacing computational models with real experimental facilities at the component and whole building level.
5. Support for hierarchical model composition to allow managing the complexity of large systems.

6. Use of a model connectivity framework that allows a model builder to assemble models in a similar way to what an experimenter would do on a workbench (cf. the *object-oriented modeling paradigm* summarized in Cellier 1996).
7. Use of symbolic algebra tools to reduce the dimensionality of the coupled systems of equations that need to be solved for simultaneously.
8. Use of numerical solvers that can solve stiff differential equation systems (which require implicit solvers with adaptive step size, cf. Hairer and Wanner 1996) that may contain boolean variables.

This list illustrates that such a computational environment needs to *simultaneously* satisfy new requirements with regard to graphical modeling environments, modeling language, symbolic and numerical methods, and code translators (that convert a model description into an executable program). It is not likely that these requirements can be met by an incremental evolution of an existing building simulation program, which typically contains hundreds of thousands of lines of procedural code that mix program statements describing the physics with program statements for control algorithms, data management and numerical solution methods. Such program code does not allow use of code generators that use symbolic algebra for reducing the dimensionality of the coupled equation systems, for automatic differentiation and for index reduction of differential algebraic equations. It also makes it impractical to use modern solvers that analyze equation systems for events and differentiability. Both measures are used in modern system simulation programs to increase computational efficiency and robustness. To realize a modeling and simulation environment that can meet the above needs, we believe it is most efficient to start with a new approach that builds on the latest advances in system modeling and simulation. However, such a new environment for modeling and simulation can still be used in conjunction with existing building simulation programs using co-simulation, for example using the Building Controls Virtual Test Bed (Wetter and Haves 2008).

Clearly, the development of such a computational environment requires expertise from various research disciplines such as computer science (for language design and code generation), mathematics (for symbolic and numerical methods) and engineering (for creating modeling libraries). Hence, a tool for such systems should not be developed by the building simulation community in isolation, but rather together with other industrial sectors to share resources. This is the approach that is followed by the Modelica consortium which has been developing the Modelica modeling language since 1997. Modelica is an equation-based, object-oriented language for modeling of systems that are described by algebraic equations, differential equations, and difference equations, and that may contain real variables, integers, boolean variables and strings (Mattsson and Elmqvist 1997).

Models written in the Modelica language cannot be executed directly. Rather, a simulation environment translates a Modelica model into an executable program. Several commercial and freely available modeling and simulation environments for Modelica that support textual and graphical modeling exist. IDA ICE 4.0, to be released in spring 2009, appears to be the first building simulation program that will support Modelica.[1] For a list of Modelica modeling and simulation environments, see `http://www.modelica.org/tools`. While the performance and price of the different tools vary, there has been significant progress in the development of these tools over the last few years, and significant investments have been made in Modelica. For example, in three European ITEA projects (EUROSYSLIB, MODELISAR and OPENPROD), about 54 Million Euros for 370 person years are invested to further develop Modelica, Modelica tools, Modelica libraries and related technology.[2]

However, what is missing in Modelica is a comprehensive library for building energy and control systems. Thus, LBNL started an open-source development effort with the aim of filling this gap. While the library is currently used within LBNL projects, it is our intention to broaden the development effort and collaborate with other developers to create an open-source Modelica library that meets the need for simulation-based innovation in building systems. This paper presents the current design of the library, which is available free of charge, including its source code, from `http://simulationresearch.lbl.gov`.

TERMINOLOGY

To facilitate the discussion of our model library, we will first introduce some terminology. For a more detailed discussion see Tiller (2001) and Fritzson (2004). In Modelica, a general object is called a *class*, which is typically restricted by the model developer. Frequently used restricted classes are a *model*, a *connector*, a *block* and a *function*. (There are other class restrictions, but these will suffice for our discussion.) A *model* typically contains time-dependent variables and parameters, which are time-independent. An equation section is used to declare algebraic and differential equations that relate parameters, variables and their time derivatives. The equations are acausal and a Modelica translator sorts and in-

[1] See `http://www.equa.se/news/2008_16.html`.

[2] See `http://www.modelica.org/publications/newsletters/2009-1`.

verts them when generating executable code. To expose interface variables, a model can contain instances of a restricted class called a *connector*. Connectors cannot contain equations. For example, the Modelica Standard Library 3.0 defines a connector for a heat port, which has variables for temperature and heat flow rate. Similarly, a connector for an analog electrical port contains variables for voltage and electrical current. These connectors declare the variables for heat flow rate and current as *flow* variables, which will cause a model translator to automatically impose conservation equations when multiple connectors are connected with each other. In contrast to a model, a *block* requires the causality of its variables to be declared. Blocks are typically used to model signal flows such as in a control algorithm. Modelica *function* objects map inputs into outputs and contain an *algorithm* section with procedural code. Functions cannot have memory and they cannot contain differential equations. Functions can be recursive, and they can call other functions that may be implemented in Modelica, C or Fortran. A model, connector, function or block can be declared to be *partial*. Partial classes cannot be instantiated. The partial keyword is typically used to force a model developer to provide a complete implementation before instantiating the class. For example, the Standard Modelica Library implements for one-dimensional heat transfer elements the partial model `Element1D` that defines two heat port connectors called `port_a` and `port_b`, variables for ΔT and $\dot{Q}$ and the equations $\Delta T = T_a - T_b$, $\dot{Q}_a = \dot{Q}$ and $\dot{Q}_b = -\dot{Q}$ where the subscripts refer to the port names. The model is declared partial because the equation that relates the temperatures with the heat flow rate is not declared at this level of the object inheritance as it is different for heat conduction, radiation or convection. To group similar classes, classes are stored in a *package* in a tree-like hierarchy. For example, the Modelica Standard Library contains the package `Modelica.Electrical` which contains the packages `Analog` and `Digital`.

USERS AND DEVELOPERS

Users of the `Buildings` library can loosely be classified into model users, model developers, and library developers.

Model users will typically graphically compose system models using models that are already available in the `Buildings` library, the `Modelica_Fluid` library (Casella et al. 2006) and the Modelica Standard Library. For model users, we are working towards creating a comprehensive set of component models that will allow modeling a variety of building energy and control systems.

Model developers will typically copy and modify existing component models, using a graphical and textual editor, or they may implement new models by using object-inheritance of an existing model. For model developers, the `Buildings` library contains partial models that implement basic functionalities, such as access to states at the component ports or conservation equations for the fluid streams, with a variable, say `Q_flow` for a heat input into a medium, which a model developer needs to assign when implementing a model. Using such a partial model, a model developer can implement a complete component model with a small set of equations. For example, an ideal heater or cooler with no flow friction is completely defined by the code[3]

```
model HeaterCoolerPrescribed
 extends Fluid.Interfaces.
  PartialStaticTwoPortHeatMassTransfer;
 parameter Modelica.SIunits.HeatFlowRate
  Q_flow_nominal
   "Heat flow rate at u=1";
 Modelica.Blocks.Interfaces.RealInput u
   "Control input";
equation
 Q_flow    = Q_flow_nominal * u;
 mXi_flow = zeros(Medium.nXi);
end HeaterCoolerPrescribed;
```

Library developers will typically develop the base models that can be used by model developers, such as `PartialStaticTwoPortHeatMassTransfer` in the example above. For the `Buildings` library, basic models of the `Modelica_Fluid` library have been used and customized for buildings applications. Developing base models requires a comprehensive understanding of Modelica and of the application domain to ensure that the models will be computationally efficient and have a high degree of reusability. Reusing modeling concepts from `Modelica_Fluid` allowed us to implement the `Buildings` library using the best practices that have been developed over the last six years by the `Modelica_Fluid` working group. By providing the partial models, ready-to-use base classes are provided to model developers so they can focus on higher level model implementations.

ARCHITECTURE

When browsing the model library, a user is exposed to the class package view. To implement new models, the object-inheritance view is also of importance to understand what models can be reused. After a short discussion of the `Modelica_Fluid` library on which our library is based, we will describe both views.

[3] For brevity, annotations have been omitted.

```
Controls --  Continuous
             Discrete
             SetPoints
Fluid    --  Actuators -- Dampers
                          Motors
                          Valves
             Boilers
             Chillers
             Delays
             HeatExchangers
             MassExchangers
             Media
             MixingVolumes
             Movers
             Sensors
             Storage
HeatTransfer
Utilities -- Diagnostics
             IO
             Math
             Psychrometrics
             Reports
```

Figure 1: Package structure of the `Buildings` library. Only the major packages are shown.

`Modelica_Fluid` Base Library

The `Modelica_Fluid` library contains component models for one-dimensional thermo-fluid flow in networks of pipes. Version 1.0, on which our `Buildings` library is currently based, was released in January 2009. It is intended to become part of the Modelica Standard Library. It provides models that demonstrate how to implement fluid flow component models that may have flow friction, heat and mass transfer. The models demonstrate how to deal with difficult design issues such as connector design, handling of flow reversal and initialization of states in a computationally efficient way. While many models of this library can be used for our application domain, we provide in the `Buildings` library models that reuse and augment models from `Modelica_Fluid` where applicable.

Packages of the `Buildings` Library

The `Buildings` library is organized into the packages shown in Fig. 1. Components in these packages augment components from the Modelica Standard Library and from the `Modelica_Fluid` library.

The package `Controls` contains models of controllers that are frequently used in building energy systems. The package `Fluid.Actuators` contains models of valves and air dampers, as well as of motors that can be used in conjunction with the actuators. In `Fluid.Delays`, there is a transport delay model that can be used in fluid flow systems. A dynamic boiler model is in `Fluid.Boilers` and different heat and mass exchanger models can be found in `Fluid.HeatExchangers` and in `Fluid.MassExchangers`. Various medium models are implemented in the package `Media`, such as for dry air, moist air and water. These medium models augment the medium models that are already available from `Modelica.Media`. Fan and pump models are stored in `Fluid.Movers`. Sensors that can be connected to a fluid stream are stored in `Fluid.Sensors`. The package `Fluid.Storage` contains models of stratified storage tanks. The package `Utilities` contains psychrometric models and blocks to format and print results to files. In the future, an interface will be added that allows linking Modelica models to the Building Controls Virtual Test Bed (Wetter and Haves 2008), and hence to EnergyPlus.

Most packages include a package called `Examples`. The example files in these packages are used to illustrate the model use and to conduct unit tests. Currently, there are around 60 example files.

Class Inheritance

We will now explain how some models are implemented in the library. While a comprehensive explanation of the whole library implementation is outside the scope of this paper, we include this section to illustrate how object-oriented modeling allows reusing the same base classes for various model implementations. While using object-oriented class definitions requires more planning when designing a library, it provides the following advantages:

1. The same code is used in many models which makes it more likely to detect model errors.
2. Code is easier to maintain since features that are shared by different models can be declared once and propagated by object-inheritance, as opposed to being copied into different source code sections.
3. Complex models can be implemented using a series of models of increasing complexity. This facilitates conducting unit tests for isolated model features, thereby increasing the chance to detect model errors earlier when they are easier and cheaper to fix.
4. Connectors and variables of similar models share the same name if they are declared in a common base class. This facilitates post-processing of simulation results. For example, because of object-inheritance, a user knows that a flow resistance element always has a public variable `dp` that reports the pressure drop.
5. Inside a system model, component models can be constrained to belong to a certain base class. They can

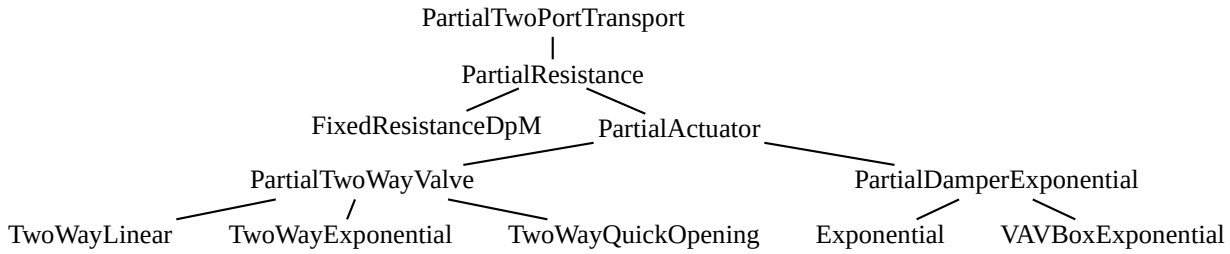


Figure 2: Object-inheritance for pressure drop elements with two fluid ports.

then be redeclared to assign an instance of a particular model inside a system model. This allows treating instances of component models in a similar way to a parameter, thereby allowing changing the behavior of a model. For example, a model for heat transfer in a wall can be propagated into a building heat transfer model, thereby allowing the creation of a building model with different model structure as described in Wetter (2006).

We will now illustrate how object-inheritance was used to implement two-way valves and air dampers. Figure 2 shows the object-inheritance tree. For the base model, we used the partial model `PartialTwoPortTransport` from the library `Modelica_Fluid`. This partial model can be used to implement models that transport a fluid between two ports while conserving enthalpy, mass and species concentration. It defines two instances of a fluid port which are called `port_a` and `port_b`. It also defines a variable that requires a model user to declare with what medium this model is used (such as dry air, moist air or water). The model also implements the enthalpy balance as $0 = \dot{H}_a + \dot{H}_b$, the mass flow rate balance $0 = \dot{m}_a + \dot{m}_b$, the species flow rate balance $0 = \dot{m}_{X,a} + \dot{m}_{X,b}$ and the pressure balance $\Delta p = p_a - p_b$. Note that how Δp is computed as a function of the flow rate is not yet specified, since the equation will be different for different models.

Next, there is a model called `PartialResistance`. This model implements a function that computes the mass flow rate as a function of pressure drop, $\dot{m} = f(k, \Delta p)$. The function $f(\cdot,\cdot)$ is an approximation to $\dot{m} = \text{sign}(\Delta p)\, k \sqrt{|\Delta p|}$ with regularization near zero for numerical reasons and to capture the laminar flow region. How k is computed is not specified at this level of the object-inheritance tree.[4]

There are two different models for specifying k. The model `FixedResistanceDpM` is a model for a fixed flow resistance in which the user can specify the point on the curve that relates mass flow rate with pressure drop. Given a nominal mass flow rate $\dot{m}_0$ and a corresponding pressure drop Δp_0, the model assigns $k = \dot{m}_0 / \sqrt{\Delta p_0}$. There are also parameters that allow a model user to specify where the transition between turbulent and laminar flow occurs. In contrast to this model, the model `PartialActuator` does not define how k is computed, because different actuators require different equations. Instead, it simply instantiates a connector for an input signal whose value is equal to the actuator opening, with $y = 0$ defined as closed and $y = 1$ defined as open.

Next, the model `PartialActuator` implements a partial model for a damper, i.e., the model `PartialDamperExponential`, and a partial model for a two-way valve, i.e., `PartialTwoWayValve`. The model `PartialTwoWayValve` defines that a valve implementation needs to specify a flow function $\phi(y) = k(y)/k(y=1)$ that relates the valve opening y with the actual flow coefficient $k(y)$ and the flow coefficient for a fully open valve, $k(y=1)$. It also specifies a parameter for the valve leakage l, i.e, $l = k(0)$ so that $\phi(0) = l/k(y=1)$.

All these partial models are stored in packages called `BaseClasses` that a typical model user does not need to browse when assembling a system model.

Next, there is a package called `Valves` with the model `TwoWayLinear` which implements the linear characteristics $\phi(y) = l + y(1-l)$, and the models `TwoWayEqualPercentage` and `TwoWayQuickOpening` that implement valve opening characteristics for equal percentage and for quick opening valves. There is also a package called `Dampers` that implements models for an air damper and a variable air volume flow box with exponential damper opening characteristics based on the partial model `PartialDamperExponential`. For example, the implementation of the two-way valve with linear opening characteristics is as follows:

```
model TwoWayLinear "Two-way valve with
  linear flow characteristics"
  extends BaseClasses.PartialTwoWayValve;
equation
```

[4] We used mass flow rate instead of volume flow rate as this leads to simpler equations. However, it would be easy to implement a model in which a user can specify the volume flow rate instead of the mass flow rate.

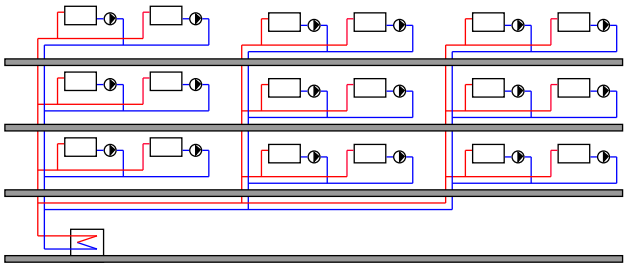


Figure 3: Schematic view of the DP system. Each radiator has a pump in its return pipe.

```
5   phi = l + y * (1 - l);
6 end TwoWayLinear;
```

For brevity, the documentation has been omitted in the above code. The documentation is html formatted text that can be translated into a documentation that displays textual documentation together with the Modelica code.

Similar object-inheritance trees are used to implement other models such as for three-way valves, for heat exchanger models and for measurement sensors.

APPLICATION

We will now show simulations that compare a conventional hydronic space heating system with thermostatic radiator valves (TRV system) to a hydronic space heating system with decentralized pumps at each radiator (DP system). The DP system is similar to the system Geniax, which the company Wilo presented to the European market in March 2009. Wilo reports that promises of the Geniax system include about 20% reduction in heating energy use and faster room temperature change during and after night setback. Fig. 3 shows schematically the DP system, with a pump at each radiator outlet. The TRV system has the same configuration, except that there is one central circulation pump at the boiler outlet, and thermostatic radiator valves are used for each radiator instead of the radiator pumps. We modeled both systems using the Modelica libraries `Buildings 0.5.0`, `Modelica_Fluid 1.0` and the `Modelica Standard Library 3.0`. The models were built and simulated in the Modelica modeling and simulation environment Dymola 7.1.

Our system was a model of a hydronic heating system of a building with three floors. Three vertical distribution pipes served 18 radiators. All mass flow rates were computed based on the pressure distribution in the piping network, which depends on the pump curves, the flow friction in the individual branches and the pump speed. All pumps had variable frequency drives that can reduce the pump speed to one third of the nominal speed. Below that value, the pumps were switched off. The heat losses of the rooms were modeled using a finite volume method to solve for the transient heat conduction through walls and floors, which we selected to be lightweight constructions. There was also steady-state heat transfer to the outside to account for heat losses due to ventilation and heat conductance through the window. In every other room, we added convective and radiative heat gains during the day to resemble people and solar heat gains. The room air was modeled as completely mixed with one state variable.

In the TRV system, each radiator had a thermostatic valve with a proportional band of 0.5 K. The boiler set point was computed as a function of the outside temperature, using a heating curve with night setback that corresponds to a reduction of the room temperature from 20 °C to 16 °C. In the early morning, the heating curve was increased to allow faster recovery from the night set back temperature. The centralized pump had a variable frequency drive that regulates the pump head.

In the DP system, each radiator had a pump that varied its speed to draw as much water as needed for tracking the room temperature setpoint. The control sequence specification was not available from the manufacturer. Based on the available literature (Baulinks 2009), we implemented the following control algorithm. A proportional controller determined the speed for each radiator pump based on the current room temperature control error. The room set point was 20 °C during the day and 16 °C during the night. To keep the boiler temperature as low as possible (for example to maximize the efficiency of a condensing gas boiler), the boiler temperature setpoint was adaptive based on the room temperature control error.

In both systems, the boiler temperature setpoint was tracked using a P-controller with hysteresis. The hysteresis was used for switching the boiler on and off. The boiler switches off if the output signal y of the boiler controller is $y < 0.3$. If $y > 0.5$, the boiler switches on and then modulates between $0.3 \leq y \leq 1$. A time relay was used to avoid excessive short cycling at very low load. All circulation pumps could reduce their speed to 30% of the nominal speed. Below this threshold, the pump switched off and remained off until its controller requested 50% of the nominal pump speed.

Fig. 4 is a view of a subset of the system model as displayed by the graphical model editor of Dymola. Each icon encapsulates a model, which may encapsulate additional models to enable a hierarchical model definition. In Fig. 4, on the left are input signals for the room temperature setpoint and the outside air. Next, there are vertical lines to connect fluid ports at the bottom and top of the floor. (For the top floor of the building, the model translator will set the mass flow rates in these pipe segments to zero, as the top ports are not connected.) In the left

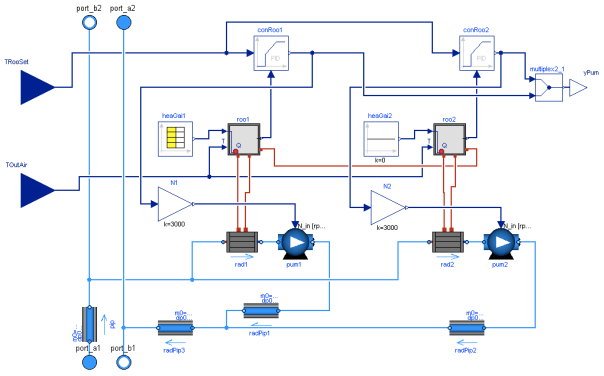
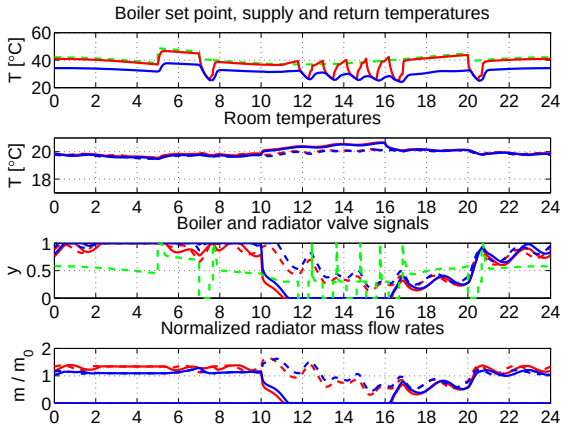


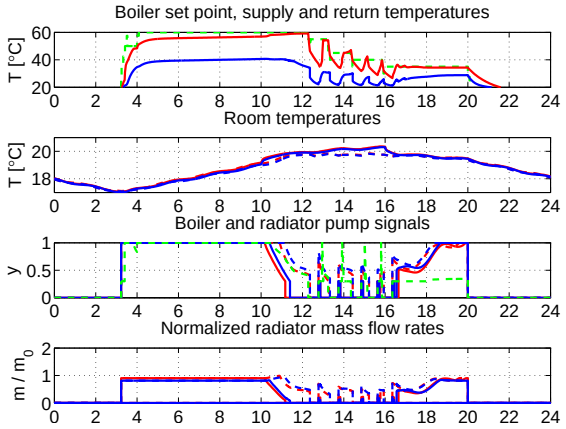
Figure 4: View of the two-room model in the graphical model editor of Dymola.

pipe, we placed a model that computes flow friction. The grey boxes in the fluid lines are finite volume models for the radiators. To the right of the radiators are the circulation pumps, and on top of the radiators are the room models. The room models contain finite volume models for computing the transient heat flow through the building constructions. Input to the room models are the outside temperatures and heat gains. The heat gains were defined by a time table for the left room, but they were set to zero for the right room. The red connection lines connect the room models to the radiators. They equate the temperatures and balance the convective and radiative heat flows, respectively, between radiator and rooms. There is also a heat flow connection between the rooms for interzonal heat transfer. Above the room models are the pump controllers. This two-room model is then instantiated nine times to form a three-storey house with three vertical distribution lines, and the distribution lines are connected to a plant model that contains the boiler and the centralized system controller. The total system model is composed of 2400 component models that form a differential algebraic equation system with 13,200 equations. After the symbolic manipulations, there were 8700 equations with 300 state variables. Building the system models for the TRV and the DP systems, including the models for the room, the radiator, the boiler and a first version of the controllers, took about a week of labor.

Fig. 5 shows the trajectories computed by the two system models. In the TRV system, the radiator valves open at night since the room temperature falls below their set point temperatures of 20 °C. This causes the radiators to release heat to the room, although at a lower rate because of the lower supply water temperature. However, in the DP system, the radiator valves and the boiler switch off while the room temperature is above the night setback



(a) TRV system



(b) DP system

Figure 5: Comparison of the dynamic system response of the TRV and DP systems for a lightweight building. The lower three subfigures show the trajectories of the four rooms that are closest and farthest away from the boiler, with the solid lines corresponding to the rooms with heat gains.

temperature, which causes a larger reduction in room temperature at night.

CONCLUSIONS

Model-based system-level analysis of the dynamic performance of building energy and control systems promises to reduce both research and development expenditures and time to market of new systems. Such a research and development process requires a flexible modeling and simulation environment that allows users to rapidly add new models of physical equipment and of continuous and discrete time controls. We showed how object-oriented equation-based modeling allows addressing some of the requirements that model-based system-level analysis imposes on the modeling and simulation environment. To better support this process, we started the development of a library of component models for building energy and control systems. The models are developed using Modelica, an open-source modeling language that has considerable support in the system-simulation community, as well as in various industrial sectors. This broad support allows sharing resources for the development of tools that are common across many engineering domains, as well as sharing domain-specific models within the building simulation community.

We discussed the software architecture of our open-source Modelica library of component models for building energy and control systems. We also demonstrated how the models can be used to compare the dynamic performance of a hydronic heating system, with circulation pumps at each radiator, to a conventional hydronic heating system with thermostatic radiator valves. Modeling both hydronic systems, including implementing dynamic models for a boiler, a radiator and a simplified room with transient heat conduction took about a week of labor. This is considerably shorter than it may have taken with many conventional building simulation programs, as modeling pressure driven flows and testing different local loop and supervisory control algorithms are often outside their capabilities.

Technical challenges remain, however, in the numerically efficient and robust simulation of such systems, and in the creation of libraries with robust models. These items are the subjects of future research and development.

ACKNOWLEDGMENTS

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

REFERENCES

Baulinks. 2009, February. Wilo startet in ein ”neues Zeitalter der Heizungssysteme”. http://www.baulinks.mobi/news/2009/0204.htm.

Casella, Francesco, Martin Otter, Katrin Proelss, Christoph Richter, and Hubertus Tummescheit. 2006, September. “The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks.” Edited by Christian Kral and Anton Haumer, *Proc. of the 5-th International Modelica Conference*, Volume 2. Modelica Association and Arsenal Research, Vienna, Austria, 631–640.

Cellier, François E. 1996. “Object-Oriented Modeling: Means for Dealing With System Complexity.” *Proceedings 15th Benelux Systems and Control Conference*. Mierlo, The Netherlands, 53–64.

Fritzson, Peter. 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons.

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

Mattsson, Sven Erik, and Hilding Elmqvist. 1997, April. “Modelica – An international effort to design the next generation modeling language.” Edited by L. Boullart, M. Loccufier, and Sven Erik Mattsson, *7th IFAC Symposium on Computer Aided Control Systems Design*. Gent, Belgium.

Tiller, Michael M. 2001. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publisher.

Wetter, Michael. 2006, September. “Multizone Building Model for Thermal Building Simulation in Modelica.” Edited by Christian Kral and Anton Haumer, *Proc. of the 5-th International Modelica Conference*, Volume 2. Modelica Association and Arsenal Research, Vienna, Austria, 517–526.

Wetter, Michael, and Philip Haves. 2008, August. “A modular building controls virtual test bed for the integration of heterogeneous systems.” *Proc. of SimBuild*. IBPSA-USA, Berkeley, CA.