Book chapter published in "Building Performance Simulation for Design and Operation," 2011, Jan L. M. Hensen and Roberto Lamberts (editors), Routledge, UK, ISBN: 978-0-415-47414-6.

A View on Future Building System Modeling and Simulation

Michael Wetter
Lawrence Berkeley National Laboratory
Simulation Research Group
One Cyclotron Road
Berkeley, CA 94720

Scope

This chapter presents what a future environment for building system modeling and simulation may look like. As buildings continue to require increased performance and better comfort, their energy and control systems are becoming more integrated and complex. We therefore focus in this chapter on the modeling, simulation and analysis of building energy and control systems. Such systems can be classified as heterogeneous systems because they involve multiple domains, such as thermodynamics, fluid dynamics, heat and mass transfer, electrical systems, control systems and communication systems. Also, they typically involve multiple temporal and spatial scales, and their evolution can be described by coupled differential equations, discrete equations and events. Modeling and simulating such systems requires a higher level of abstraction and modularisation to manage the increased complexity compared to what is used in today's building simulation programs. Therefore, the trend towards more integrated building systems is likely to be a driving force for changing the status quo of today's building simulation programs. This chapter discusses evolving modeling requirements and outlines a path toward a future environment for modeling and simulation of heterogeneous building systems.

A range of topics that would require many additional pages of discussion has been omitted. Examples include computational fluid dynamics for air and particle flow in and around buildings, people movement, daylight simulation, uncertainty propagation and optimisation methods for building design and controls. For different discussions and perspectives on the future of building modeling and simulation, we refer to Sahlin (2000), Augenbroe (2001) and Malkawi and Augenbroe (2004).

Learning objectives

The learning objective of this chapter is to better understand the limitations of existing building simulation programs to model and simulate the performance of integrated building energy and control systems and how recent advances can overcome these limitations. In this chapter, the reader will be exposed to (i) motivating factors that may change the way we model and simulate building energy and control systems, (ii) technologies that enable such a change, and (iii) a path for realizing a next generation modeling and simulation environment. While an in-depth discussion of the various enabling technologies is not possible in a single chapter, the reader will be exposed to the main concepts, and references to further literature are provided.

Key words

Requirements for building system simulation / modeling versus simulation / modular environments / object-oriented equation-based modeling / agent-based modeling / multi-domain modeling / management of complexity / Building Information Model

Introduction

Imagine an integrated design team sketching a building envelope in an object-oriented CAD system. A dashboard provides immediate feedback about the environmental impact and life cycle costs as different design variants are tested. After developing an initial building design, the HVAC engineer downloads the Building Information Model (BIM), i.e., an electronic representation of the building parameters, draws the HVAC schematic diagram and drags standard supervisory control sequences from a library into the schematic diagram. At the press of a button, the HVAC engineer runs a whole building energy analysis and sizes the components while taking into account the reduced component size due to exploitation of free cooling and active facade systems. Next, the HVAC engineer replaces the simplified component models with refined equipment models downloaded from an electronic catalogue. The models in this catalogue have been used by the manufacturer during the product development and can represent the physical behaviour at different levels of temporal and spatial resolution; simplified performance models are used for annual energy analysis while detailed models may later be used during operation to determine a chiller's refrigerant charge in a fault detection and diagnostics program. After the HVAC engineer specifies the equipment, she/he uploads the data to the BIM that will be submitted to code authorities for performance-based code compliance checking within minutes and to generate bidding materials and construction plans. During building operation, the BIM data will be integrated with models and algorithms for optimisation-based control, fault detection, diagnostics and preventive maintenance.

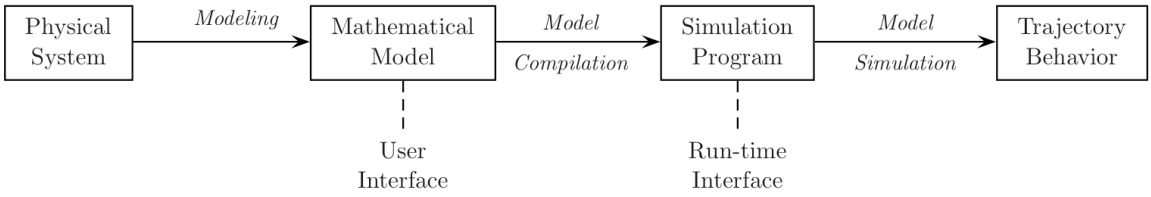


Figure 1: Modeling vs. simulation. (Adapted from Cellier and Kofman (2006), with kind permission of Springer Science+Business Media.)

What building simulation program will support such a life-cycle approach to simulation from product R&D to building operation? This is a focus of this chapter. Central to our discussion is the belief that the next generation tools that interface with the workflow of simulationists are not building simulation programs in the conventional sense. To explain this, we use the illustration from Cellier and Kofman (2006), shown in Figure 1, which distinguishes between the *mathematical model* and the *simulation program*. Cellier and Kofman state that the whole purpose of a mathematical model is to provide the human user of the modeling and simulation

environment with a means to present knowledge about the physical system in a way that is as convenient to the user as possible. This may be for a building envelope an object-oriented CAD system, for an HVAC system a graphical schematic editor and for control systems a block diagram and finite state machines. Thus, the mathematical model represents the user interface. The mathematical model does not depend on how it is going to be used by the simulation program. The primary purpose of the simulation program is to compute the system response, typically in the form of time trajectories. The simulation program is automatically generated from the mathematical model. It typically provides a run-time interface with which the user can control the simulation process. Such a separation between mathematical model and simulation program is a critical underlying principle for a future modeling and simulation environment for building systems as we will see in this chapter. By following this principle, it is possible to structure the problem more naturally in the way a human thinks, and not how one computes a solution.

Motivating factors for a future modeling and simulation environment

To increase performance and improve comfort, building systems are becoming increasingly integrated. Integration is achieved physically by using systems that allow better energy recovery, energy storage and increased use of ambient energy sources and sinks for heating, cooling, ventilation and lighting. Integration is also achieved operationally by means of integrating controls for facade, lighting, HVAC, combined heat and power, solar systems, electrical grid and security. The characteristics of these systems are that they involve multiple functional domains (heat and mass transfer; fluid dynamics; daylighting; electricity generation and conversion; controls; communication across networks), and their temporal evolution may be described in the continuous time domain, in the discrete time domain and in a discrete event domain. Typical time-scales range from the order of subseconds (communication networks) to years (geothermal heat sources). This heterogeneity poses new challenges for modeling and simulation, which are used to reduce time-to-market during research and development, and to improve performance and cost-effectiveness during the design of innovative building energy systems and associated algorithms for controls, fault detection and diagnostics.

Due to the increased importance of controls to improve building performance, the control representation in building simulation programs needs to resemble more closely how actual control algorithms are implemented. Given the different topologies in which building systems can be configured, such as using a double facade as a ventilation air exchange path or using a thermally activated ceiling slab for thermal storage, a future modeling environment needs to allow a more intuitive representation of such complex systems that allows a user to quickly add new component models and reconfigure components to form new systems. Furthermore, the current separation between model developer and model user is not conducive to stimulate innovation.

Given the increased integration of different domain disciplines, a future environment for modeling and simulation should also enable the concurrent collaboration of multi-disciplinary teams, as opposed to a single person, and the sharing of models to integrate different domains. This may become increasingly important because much of the innovation in building science is likely to happen at the interface between different disciplines; for example, building physics for passive energy storage, mechanical systems for efficient energy conversion, controls systems for reducing irreversibilities (fluid flow friction and heat transfer), communication systems for

propagation of sensor and actuator signals and mathematics to extract better quality information from the vast amount of data measured by building automation systems. Addressing the need for different discipline experts to collaborate more effectively will require different design of future tools.

Because embedded computing is becoming cheaper and more powerful and the performance of low-energy buildings is sensitive to controls and equipment degradation, it is likely that computational models will be used for commissioning, fault detection, diagnostics and model-based controls. In this context, the use of models can range from an individual piece of equipment, such as for a chiller fault detection, or to whole communities or estates that may be connected by a micro-grid for electricity, heating and cooling. Since sustainable buildings will have a lifespan that exceeds multiple generations of programming languages, operating systems and building simulation programs, models used during operation should be expressed at a high enough level of abstraction that ensures maintainability, portability, and facilitates upgrading to newer technology as buildings are retrofitted.

With respect to using building simulation for training of service technicians, commissioning agents and building operators, today's building simulation programs fall short in teaching how mechanical systems and their control systems operate. At present, models of these systems are often so simplified that they do not capture the dynamic behaviour and part-load operation of the mechanical system or the response of feedback control systems. Also, in many building simulation programs, the governing physical equations cannot be readily inspected by the user, which renders them as a black-box. Immersive simulation has the potential to improve the knowledge about the operational behaviour of buildings by presenting results of building simulation in a form that is closer to the human than a graph on a computer screen. Work to date has been primarily focused on computation and visualisation of spatial distribution of temperature, flow and comfort in rooms (Malkawi, 2004; Wenisch et al., 2005). A versatile environment that would allow a realistic modeling and simulation of the mechanical and control system, and that could be reconfigured by the user to examine a variety of systems, would extend the promise of immersive simulation to training professionals in the operation of building systems.

The transition towards integrated systems introduces new challenges. Effective modeling strategies need to manage the increased complexity, the increased fragmentation of the supply chain and the need for working in multidisciplinary design teams using shared models. In the electronics design automation community, which faces challenges that are similar to the ones of the building industry, there are calls for a new design science to address system-level design challenges (Sangiovanni-Vincentelli, 2007). The basic tenet of the advocated design flow is the identification of design as a meet-in-the-middle process, where successive refinements of specifications meet with abstractions of potential implementations. The refinement and abstraction process takes place at precisely-defined layers. It remains to be shown how this design approach can best benefit the building industry and what its implications are for building system models. The embedded systems community also called for a new system science that is jointly physical and computational in order to address challenges posed by networking physical devices that contain embedded control systems (Lee, 2006). In this community, the semantic of a model is expressed at a high enough level of abstraction to allow formal analysis of system properties and to generate code automatically that can be used for design, synthesis and operation. It would be a surprise if advances in these related disciplines would not also cause the field of building simulation to embrace a higher level of abstraction that better addresses challenges in research,

design and operation of integrated building systems.

A closer look at the software architecture of today's building simulation programs reveals that they are ill-suited to address the above needs. Many building simulation programs do not exploit advances in computer science that allow systems and their evolution to be expressed at a higher level of abstraction than that of imperative Fortran subroutines. Most of today's building simulation programs are large, monolithic programs that intertwine physics equations, statements that control the program flow logic, numerical solution methods, and data input and output routines. This makes extending these programs for applications that were not envisioned by their developers difficult. Lack of modularity has forced developers of building simulation programs to try to catch up by adding modeling capabilities that allow simulating new technologies. However, it may be more beneficial to develop a modular platform that can rapidly be extended and customised by users to drive the innovation of new systems and to integrate simulation better into building design and operation. In view of the gap between current building simulation programs and modern modeling and simulation environments, we will now take a step back to describe some basic needs that motivate a new approach to building system modeling and simulation.

Needs for more natural system representations

"Object programs produced by Fortran will be nearly as efficient as those written by good programmers." This statement from the Fortran manual (IBM, 1956), released in 1956, shows that as early as fifty years ago, there was a trend towards languages that were closer to the problem at hand. Even then, experts questioned whether the "high-level" programming language would be adequately efficient. Today, most building simulation programs are still written using Fortran to encapsulate causal assignments in programming procedures. In these programs, procedures call other procedures, thereby transferring the locus of control from one procedure to another. Inside these procedures are imperative statements for algebraic equations, differential equations, difference equations and numerical solution algorithms. This syntax for formulating physical phenomena was motivated by the programming languages that were available decades ago. However, the syntax is far removed from how a physicist would describe the laws that relate physical quantities to each other or how an electrical engineer would describe communication in a network.

Physical systems are more conveniently modeled by (i) defining an object, (ii) exposing its boundary conditions and (iii) encapsulating inside the object mathematical constraints between the boundary conditions, the state variables and their derivatives. Consider for example a model for a heat conducting rod that stores no energy. It is not possible to determine whether heat flows through the thermal conductor because its ends are at different temperatures, or whether the temperatures at its ends are different because of the heat flow. So why should a user have to pick one or the other view when writing a model for a thermal conductor? It is more natural to create an object that exposes heat flow rate and temperature, i.e., (Q_1, T_1) and (Q_2, T_2), and encapsulates the constraints $Q_1+Q_2=0$ and $Q_1 = K (T_1 - T_2)$, where K is the heat conductance (see Figure 2). How these equations are solved should be left entirely to an algorithm that generates a simulation program.

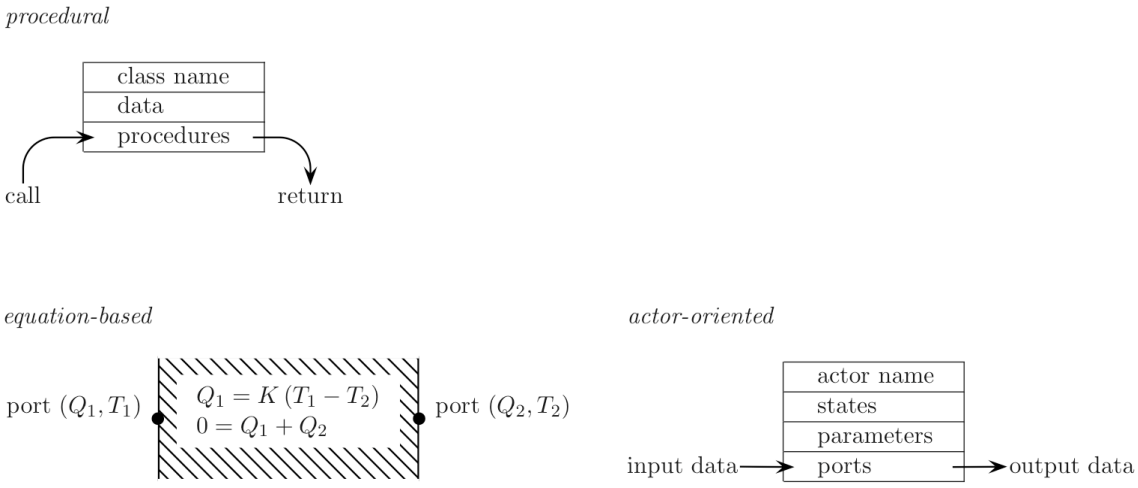


Figure 2: Comparison between procedural, equation-based and actor-oriented component models. (Procedural and actor-oriented illustration adapted with permission from Lee (2003).)

Such an equation-based, object-oriented model formulation is convenient to model physical systems that can be described by differential algebraic systems of equations. But is it also convenient to model *cyber-physical* systems? Cyber-physical systems are systems in which embedded computers and networks monitor and control physical devices with feedback loops, and the physical processes affect computations and vice versa (Lee, 2006). Cyber-physical systems are becoming increasingly prevalent in buildings, and their size, in terms of nodes that send and receive signals, is growing rapidly. Components in cyber-physical systems exchange a sequence of discrete messages with each other through wired or wireless networks, which may lose or delay some messages. The sequence of discrete messages describes the evolution of each component's state variables. Such communication networks can be conveniently modeled by *finite state machines*. Finite state machines are objects that, if given the current state and an input, return the new state and an output. For computer modeling, it is convenient to represent such systems with actor-oriented component models.

Figure 2 illustrates the disconnect between how building simulation programs are written and how the evolution of physical systems and communication systems can be described. In procedural programs, a procedure that belongs to a class is called. This procedure manipulates data, may call other procedures, and then returns to transfer the locus of control back to the calling procedure. This is how typical building simulation programs are written. In equation-based programs, classes are typically encapsulated based on thermodynamic system boundaries. Ports are used to expose potential and flow variables (e.g., temperature and heat flow rate) without specifying what is input and output as this is not needed to describe physics. Acausal relations define constraints between the port data and between the thermodynamic states of the system. In actor-oriented programs, actors contain ports that receive and send tokens. A *model of computation* (Lee, 2003) specifies operational rules that determine when actors perform internal computations, update their states and perform external communications, which can, for example, be at fixed time intervals or whenever new tokens arrive. The ports are connected to other components' ports in order to communicate with each other.

Needs to reduce software cost and development time

Cellier (1996) states the main factors that reduce cost and development time of software as follows:

Reusability: A software design methodology that ensures optimal reusability of software components is the most essential factor in keeping the software development and maintenance cost down.
Abstraction: Higher abstraction levels at the user interface help reduce the time of software development as well as debugging. The conceptual distance between the user interface and the final production code needs to be enlarged. Software translators can perform considerably more tasks than they traditionally did.

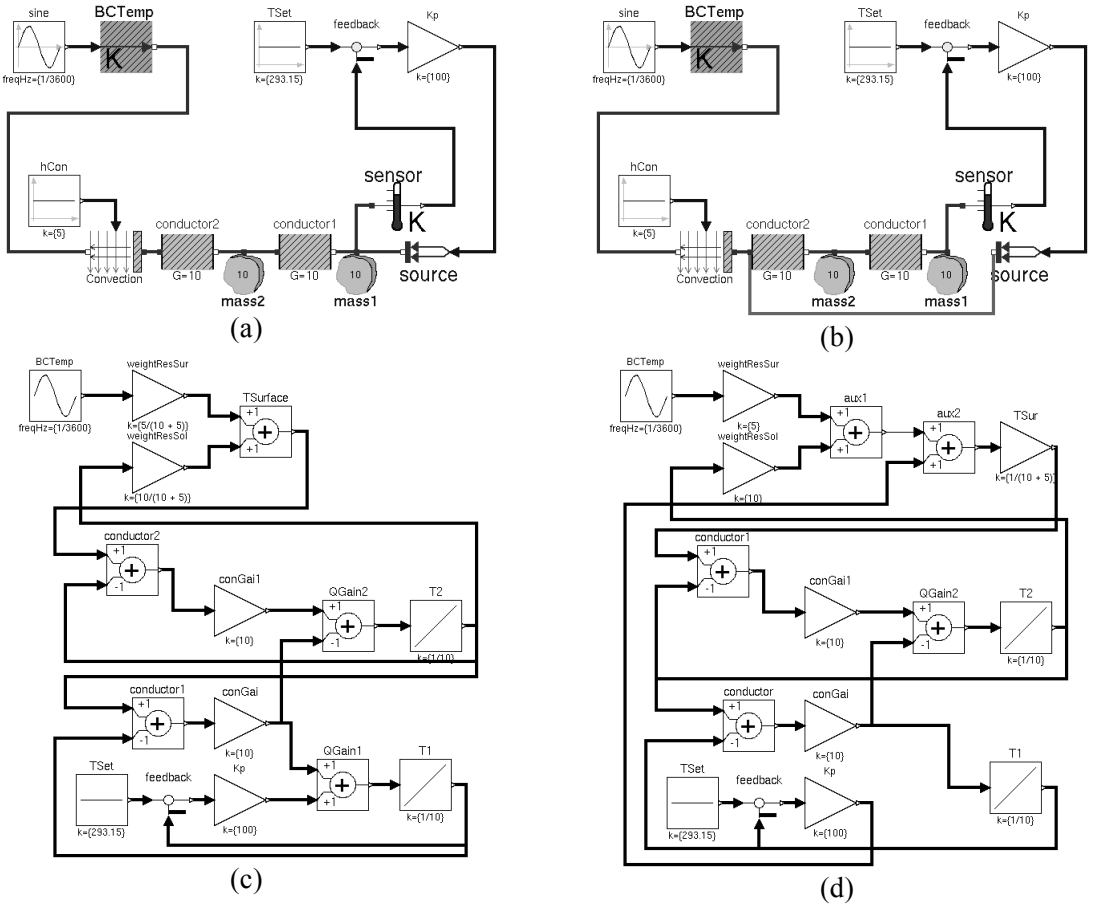


Figure 3: Representation of heat flow models with feedback control.

The reusability of code can be significantly increased through the use of equation-based object-oriented models that separate acausal relations between model variables from their solution algorithms. For example, Figure 3 shows models of a heat conducting wall in which the heat input is regulated by a feed-back control loop. Only half of the wall depth has been modeled because the wall is assumed to be symmetric and exposed to the same boundary conditions. In Figure 3(a), heat is added from the object called "source" to the wall's axis of symmetry, i.e., to the object called "mass1" which is at the centre of the wall. In Figure 3(b), heat is added to the surface of the wall. The top row shows the implementation using equation-based object-oriented models, where each icon encapsulates a model for an individual physical phenomenon (such as heat convection, heat conduction and heat storage). With this representation, the system being modeled is easily recognised. In contrast, the bottom row shows the same physical models, but they have been implemented using causal models in the form of block diagrams. Figure 3(a) and Figure 3(c) describe the same system, as do Figure 3(b) and Figure 3(d). Note that the top row also contains causal models, but only for the control system since this is most naturally represented using input/output blocks. The equation-based object-oriented system models are easier to read because the schematic model representations only defines the connectivity between the component models, but there is no notion of causality needed. In contrast, the block-diagrams in the bottom row impose causality for each component and for the information flow within the system models. This limits model readability and reuse. It also led to more work for creating the models.

When developing the equation-based object-oriented model shown in the top row, each individual component model, such as for the heat conductor or the heat storage, could be tested individually in isolation of the overall system. Such unit tests help detect possible programming errors earlier when it is cheaper to fix them. Assembling the equation-based object-oriented system model by connecting lines between the components then poses little risk for introducing errors. To assemble the causal system model, however, gains must be computed and heat flows must be added and connected to the corresponding integrator, which is time consuming and presents opportunities to make mistakes. The left and right columns of the figure differ only in the location where heat is added to the construction. For the equation-based object-oriented model, all a modeler must do is reconnect one line; for the causal model the modeler must add and delete blocks and recompute gains, thus reusing a significantly smaller part of the model and possibly introducing mistakes. The example illustrates that the equation-based object-oriented system representation not only reduces development time but also allows a more natural system representation as discussed in the previous section.

Needs to integrate tools in a coherent workflow

Researchers and practitioners represent different user groups who use modeling and simulation within different workflows and therefore have different requirements that are best addressed by a modular environment. With a modular approach, the same underlying core of the environment can be used for both user groups while customized interfaces can address the individual requirements. An example of a modular software environment is Linux, which can be customised to run in embedded systems such as cell phones, in graphical desktop environments, in server farms and in super computers. Modularisation also makes it significantly easier for users to participate in program development; an example is the modular design of TRNSYS' simulation components that allows a large user community to develop simulation components that extend the scope of the program.

Researchers require an environment that integrates well with computational-based rapid prototyping and system design processes that reduce cost and product time-to-market. The environment must also provide formal ways for managing the increased system complexity and integration challenges in heterogeneous design and supply chains (Sangiovanni-Vincentelli, 2007; Barnard, 2005). For product development it is common that simulation models are progressively replaced with more detailed models and with hardware as development progresses. Such processes typically include automatic code generation that translates a control algorithm developed in a modeling and simulation environment into code that can be downloaded to control hardware. New environments should allow users to add novel component and system models rapidly, and combine models in new system topologies to analyse new configurations for integrated building systems. Meeting these requirements mandates an environment that consists of modular models with standardised interfaces that can be individually tested and replaced with more detailed models as more information becomes available during the design, or supplanted by hardware that is linked with the simulation model.

Practitioners require that a new environment for modeling and simulation integrates well with the building delivery process. Since BIM are becoming increasingly used, mandated by large building owners, supported for energy code compliance checking and profitable for design teams to use, it is important that the building simulation program can be generated using BIM inputs. Integrating BIMs into the building life cycle will require that mathematical models are instantiated and parameterised using data specified in or referenced by a BIM. As controls become increasingly important in reducing energy consumption, peak electrical power and downsizing or even eliminating mechanical equipment (such as eliminating a chiller if night-ventilation of a building suffices to meet the comfort requirements), control-relevant information also needs to be specified by the BIM. To avoid data inconsistency between design and operation, and to avoid duplicate data entry, the environment should also be able to use a BIM to instantiate and parameterise models for model-based controls, fault detection and diagnostics. In this context, BIM is used in a broader sense than today and will include formal specifications of control sequences. The specifications will have a semantics that is expressive enough so they can be used during design to simulate the system performance, during bidding for estimating construction costs, during commissioning to verify the proper operation, and during operation for controlling the building system. Model instantiation and parameterisation is easiest to achieve if the components specified in the BIM and in the simulation model have the same modularisation and if the connectivity between components is only constrained by how physical laws permit real components to be connected. To meet these requirements, a more general approach for component connectivity rules is needed than what can be found in many building simulation programs.

Enabling technologies

The previous section showed that building systems are becoming increasingly integrated across different functional domains. System integration increases design and operational complexity. A future modeling and simulation environment for building systems will address this challenge by using technologies that enable an intuitive representation of the system topology, the intrinsic behaviour of individual components and the interaction among the components once they become part of a system. This section discusses these technologies.

We will make a distinction between *modeling* and *simulation* as introduced by Cellier and Kofman (2006) and illustrated in Figure 1. Since building energy analysis requires large amounts of input data, it is convenient for our discussion to further separate mathematical modeling into behavioural modeling and data modeling. By *behavioural modeling* we mean specifying the mathematical equations that model a phenomenon, which Polderman and Willems (2007) call the behavioural equations. Behavioural models may be based on physical laws, performance curves or by functions that update states and outputs of a finite state machine. By *data modeling* we mean representing the static data of a system without specifying the mathematical model that describes how the system evolves in time.

The purpose of modeling is to create a mathematical model from a physical system. From the mathematical model, a simulation program can be generated by using symbolic manipulations to sort equations, invert equations and replace derivative operators with numerical discretisation schemes. Generating a simulation program can be done manually by a programmer, as is the current practice in most building simulation programs such as in TRNSYS or EnergyPlus. Alternatively, it can be automated, as done in equation-based modeling environments such as Dymola, OpenModelica, SPARK, IDA or EES. The manual approach requires a model developer to write a simulation program (or subroutine), interface it with numerical solution methods, integrate it into the program kernel and implement methods to retrieve parameters and time-varying inputs and to write outputs. Manually generated programs are error prone and take considerably more time to write than specifying the actual physical relations that govern the object to be modeled. Equation-based modeling, in contrast, allows a model builder to concentrate the effort on describing the physical constraints and state transitions that define the evolution of the system. Generating efficient code for numerical simulation is left to a code generator.

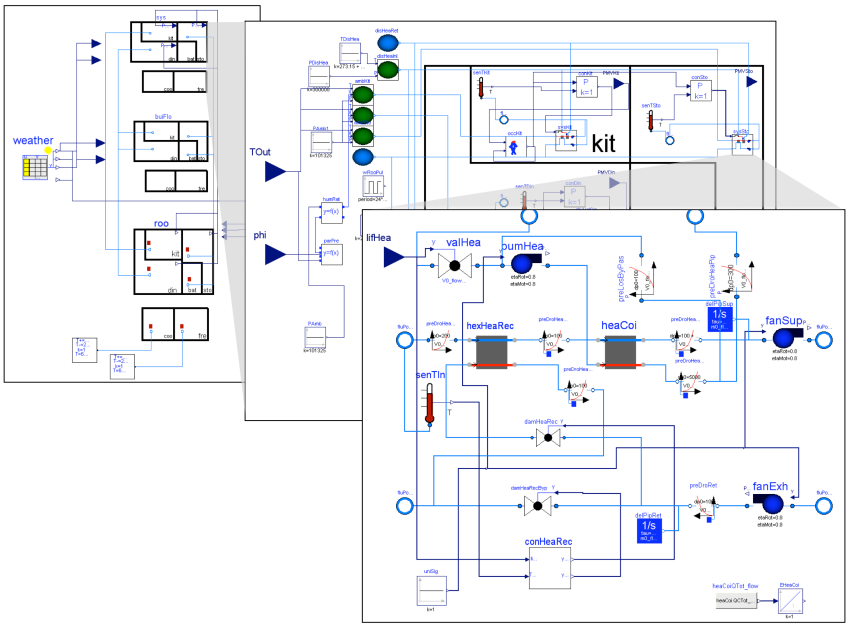
Behavioral models

In the building simulation community, the need for more flexible means of constructing simulation models was expressed two decades ago by Sowell et al. (1986). A draft research plan was assembled by an international team working at the Lawrence Berkeley National Laboratory in 1985 for the collaborative development of a next generation building simulation program called the Energy Kernel System (EKS) for building simulation. A prototype software system called SPANK (Simulation Problem Analysis Kernel) that eventually became SPARK was created that contained some of the features envisioned for EKS (Sowell et al., 1986, 1989). Another development that embraced equation-based model formulation for building simulation is the Neutral Model Format (NMF) (Sahlin and Sowell, 1989; Sahlin, 1996). NMF is the primary modeling language used in IDA, an equation-based modeling and simulation environment that is used by researchers and practitioners and thas recently been used to translate and simulate models written in the Modelica language described below (Sahlin and Grozman, 2003). The Engineering Equation Solver (EES) is another equation-based modeling and simulation environment with an interface to thermophysical property functions, such as for steam and refrigerants (Klein and Alvarado, 1992).

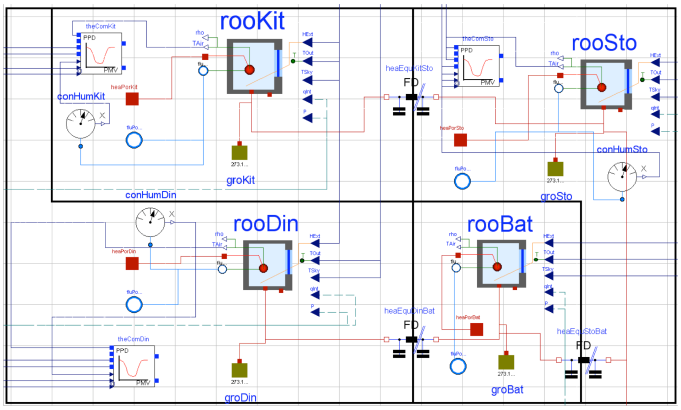
In the meantime, a significantly larger industry-driven effort has been started that includes a wider range of industrial sectors and academic partners. In 1996, a consortium formed to develop Modelica, a freely-available, equation-based object-oriented modeling language that is designed for component-oriented, multi-domain modeling of dynamic systems. The goal of the consortium

is to combine the benefits of existing modeling languages to define a new uniform language for model representation (Mattsson and Elmqvist, 1997; Fritzson and Engelson, 1998). Over the past decade, the Modelica language has gained significant adoption in various industrial sectors and has been used in demanding industrial applications. It is well positioned to become the de-facto standard for modeling complex physical or communication systems. The rise of Modelica provides an opportunity to revive the efforts of the International Energy Agency Task 22 to create a model library for building simulation, which was then implemented in NMF, that can be shared among different modeling and simulation environments (Vuolle et al., 1999). Modelica libraries for multi-domain physics include models for control, thermal, electrical and mechanical systems, as well as for fluid systems and different media (Elmqvist et al., 2003; Casella et al., 2006). Modelica models can contain differential equations, algebraic equations and discrete equations. Using standardised interfaces, a model's mathematical relations among its interface variables are encapsulated, and the model can be represented graphically by an icon. The encapsulation facilitates model reuse and exchange and allows connecting component models to system models using a graphical or textual editor. The tenets of Modelica models are that each component represents a physical device with physical interface ports, such as heat flow rate and temperature for solid ports and pressure, species flow and enthalpy for fluid ports. Defining a device- and physics-oriented system architecture enables intuitive model construction. The Modelica approach is realised by following the *object-oriented modeling paradigm*, a term that was coined by Elmqvist (1978) and summarised by Cellier et al. (1995) as follows:

Encapsulation of knowledge: The modeler must be able to encode all knowledge related to a particular object in a compact fashion in one place with well-defined interface points to the outside.
Topological interconnection capability: The modeler should be able to interconnect objects in a topological fashion, plugging together component models in the same way as an experimenter would plug together real equipment in a laboratory. This requirement entails that the equations describing the models must be declarative in nature, i.e., they must be acausal.
Hierarchical modeling: The modeler should be able to declare interconnected models as new objects, making them indistinguishable from the outside from the basic equation models. Models can then be built up in a hierarchical fashion.
Object instantiation: The modeler should have the possibility to describe generic object classes, and instantiate actual objects from these class definitions by a mechanism of model invocation.
Class inheritance: A useful feature is class inheritance, since it allows encapsulation of knowledge even below the level of a physical object. The so encapsulated knowledge can then be distributed through the model by an inheritance mechanism, which ensures that the same knowledge will not have to be encoded several times in different places of the model separately.
Generalised Networking Capability: A useful feature of a modeling environment is the capability to interconnect models through *nodes*. Nodes are different from regular models (objects) in that they offer a variable number of connections to them. This feature mandates the availability of *across* and *through variables*, so that power continuity across the nodes can be guaranteed.



(a) View of the system model and the HVAC model



(b) View of a section of the multizone building model

Figure 4: Schematic view of a building system model in Modelica that was constructed using different hierarchical layers to manage the model complexity, and that decomposes different domains. In the top figure (a), on the left, there are models for the HVAC system (top), for the interzonal air flow (middle) and for the building envelope heat transfer (bottom), which are connected to each other by fluid loops and control signals to form a coupled system of equations. The top right shows the inside of the HVAC system definition for each thermal zone, with an icon for the HVAC secondary loop, for the controller and for the user that may adjust the set points to maintain comfort. The bottom right shows the actual implementation of the HVAC secondary loop with the heat exchangers, the fans and pumps. The bottom figure (b), shows a section of the multizone building model with models for the rooms that are connected to each other with finite difference heat transfer models. Also shown are models for the thermal comfort.

Figure 4 illustrates how a multizone building model was created by graphically connecting physical ports for heat flow rate, air flow rate and control signals of thermal zone models, airflow models and HVAC system models. Each icon encapsulates a model that may encapsulate other models, thus allowing a model builder to manage the complexity of large system models through hierarchical modeling. Hierarchical model building also facilitates model reuse and testing of submodels before they are assembled as a large system model that may be difficult to debug. Object-oriented model construction with clearly defined encapsulations accelerates model development by allowing concurrent construction of multiple process models by their respective experts. For example, the HVAC and controls engineer can construct their models simultaneously and combine them later to a system model. The concurrent workflow with subsequent model combination is illustrated by Figure 4(a). On the left side of the figure, the three separate models represent the HVAC systems, the interzonal air flow and the building heat transfer (Wetter, 2006a,b). The different domain models are integrated with each other by drawing lines between the model ports. The lines automatically generate equality constraints for across variables (temperature, pressure) and conservation equations for through variables (heat flow rate, mass flow rate).

Libraries of component models that allow assembling such systems are created using objects with standardised interfaces that define all independent variables necessary to describe the desired effect. Using these standardised interfaces makes models compatible with each other without having to add converters that convert one set of interface data into another one. For example, in Modelica's thermal library there is a connector called `HeatPort` that defines an interface for elements that transfer or store heat by introducing the tuple temperature and heat flow rate as

```
1 partial connector
2   Modelica.Thermal.HeatTransfer.Interfaces.HeatPort
3      "Thermal port for 1-D heat transfer";
4   SI.Temperature T "Port temperature";
5   flow SI.HeatFlowRate Q_flow
6     "Heat flow rate (positive if flowing into the component)";
7 end HeatPort;
```

The keyword `flow` declares that all variables connected to `Q_flow` need to sum to zero. For example, if two `HeatPorts` that are called `port1` and `port2` are connected, a Modelica translator will generate the equality constraint `port1.T = port2.T` and the conservation equation `port1.Q_flow + port2.Q_flow = 0`. Similar connectors are defined for other physical interfaces including fluid ports, electrical signals and translational and rotational bodies. The above heat port connector is used to define an interface for one-dimensional heat transfer elements with no energy storage in the following partial model:

```
1 partial model Element1D
2   "Partial heat transfer element that does not store energy"
3   SI.HeatFlowRate Q_flow "Heat flow rate";
4   SI.Temperature dT "port_a.T-port_b.T";
5   HeatPort port_a;
6   HeatPort port_b;
7  equation
8   dT = port_a.T - port_b.T;
```

```
 9   port_a.Q_flow = Q_flow;
10   port_b.Q_flow = -Q_flow;
11 end Element1D;
```

This model is incomplete because it does not specify how temperatures and heat flow rate depend on each other. This incompleteness is denoted by the keyword `partial` which has two purposes: It shows to a user that the model cannot be instantiated, and it tells a Modelica translator that the model is allowed to have more variables than equations, which is used for automatic error checking. To define a complete model, the above partial model is used to define a thermal conductor as

```
1 model ThermalConductor
2   "Thermal element transporting heat without storage"
3   extends Interfaces.Element1D;
4   parameter SI.ThermalConductance G "Thermal conductance";
5 equation
6   Q_flow = G*dT;
7 end ThermalConductor;
```

The thermal conductor model is graphically represented by an icon that is used, for example, in the system model shown in Figure 3. Note that by replacing the parameter declaration on line 4 and the equation on line 6, the semantics can be changed to represent other one-dimensional heat transfer elements, such as a model for convective heat transfer.

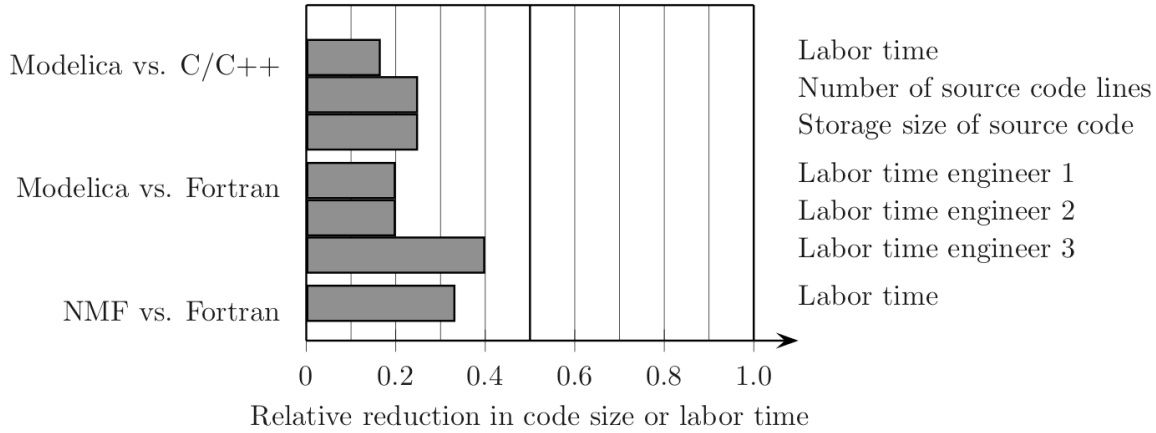


Figure 5: Comparison of relative reduction in labor time and code size for equation-based vs. procedural programming languages.

Equation-based modeling languages have been shown to significantly reduce model development time compared to conventional programming languages such as C/C++ or Fortran, as shown in Figure 5. A comparison of the development time for the BuildOpt multizone thermal building model which is implemented in C/C++ and a comparable implementation in Modelica showed a development time five to ten times faster when using Modelica (Wetter, 2005, 2006a; Wetter and Haugstetter, 2006). The Modelica development is faster primarily because Modelica is an acausal, equation-based, object-oriented language. This automated the work for writing routines

for data input and output, sorting equations and procedure calls, implementing numerical solution methods and managing the large amount of data that is involved in building simulation. The shorter development time also manifests itself in a code size four times smaller (6,000 lines instead of 24,000 lines of code, not counting freely available libraries). In another experiment, the author asked three engineers familiar with Fortran and Modelica to implement the system shown in Figure 5(a) in Fortran and in the Modelica modeling and simulation environment Dymola. Even for such a simple problem where data input, output and storage are easy to implement, the model in Modelica was built 2.5 to 5 times faster than in Fortran. Similar increase in productivity has also been reported by Sahlin (2000), who was three times faster in implementing a thermal building simulator using the equation-based language NMF. These benchmarks indicate significant time-savings for modeling when using equation-based modeling languages.

Data models

A BIM is an instance of a data model that describes a specific building unambiguously. Several data models are used by the construction industry. The only open, object-oriented data model that covers the whole building life cycle are the Industry Foundation Classes (IFC), developed by the International Alliance for Interoperability and currently under consideration at the International Organization for Standardization (ISO) to become an international standard (ISO/PAS 16739:2005). Economic motivations for using BIMs include an increase in productivity through data interoperability among the participants of the building delivery process as well as data integrity during project planning, cost analysis, energy analysis and automatic code compliance checking. This section focuses upon how a future modeling and simulation environment for building systems can make BIM use throughout the building life cycle more seamless. Therefore, our discussion is tool-oriented to show how building performance assessment tools can best address requirements that originate from the use of BIM. We refer the reader to Augenbroe et al. (2004) and Lam et al. (2004) for a process-oriented discussion of simulation models within the building process and to Bazjanac (2008) for a discussion of semi-automated data transformation between BIM authoring tools and building simulation tools.

This discussion assumes that future BIMs will contain specifications for building controls that will be used during design for building simulation as well as during operation for model-based algorithms for controls, fault detection and diagnostics. Since controls are an essential tool for increasing building system efficiency, they would be a natural extension to a BIM, and the IFC2x platform specification indeed mentions building automation as one discipline that it aims to support. To enable a seamless combination of data models represented by a BIM and behavioural models represented in an equation-based object-oriented language that can generate a building simulation program, the component modularisation and the variables that connect components should be identical in the data model and the behavioural model. Furthermore, data and behavioural models need to allow connecting components in the same way as one would connect actual components. Otherwise the set of possible HVAC systems is larger than the set of HVAC systems that can be simulated, which makes seamless BIM use difficult if not impossible for some HVAC systems.

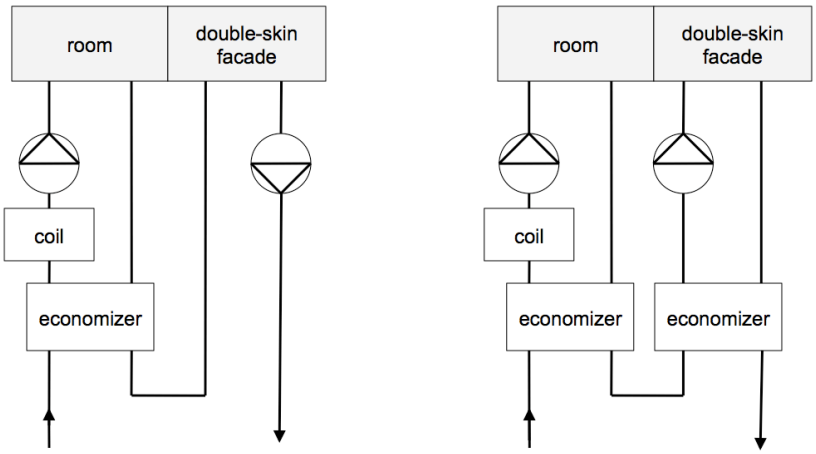


Figure 6: Schematic diagram of an HVAC system that uses a double-skin facade for the exhaust air (left side) and implementation in EnergyPlus that required two air loops (right-side). (Figure courtesy of Tobias Maile, Lawrence Berkeley National Laboratory.)

Many building simulation programs are based on software principles that do not match how buildings and energy systems are actually built and controlled. Also, their modeling language is not expressive enough to allow computing the evolution of heterogeneous systems in a way that properly accounts for concurrency of event-driven systems. This can make the use of models for simulation-based controls and commissioning difficult. In addition, the model modularisation and model connectivity rule is imposed by the program developers. They decide what should be lumped in a monolithic subsystem model, what causality should be imposed on the equations to translate them into procedural code and what connections should be required for an object. These decisions are not motivated by how a BIM represents an HVAC system and they do not necessarily reflect real HVAC systems. Rather, the selection of component modularisation, component causality and connection variables is motivated by accommodating the data flow and solution algorithms imposed by the program architecture. For example, the room model of EnergyPlus computes a heating or cooling load in units of power. This load is then sent to zone equipment units, such as a fan coil, that try to meet the load. Thus, the control input of the fan coil is a load in watts, as opposed to a signal from a thermostat. This type of component connectivity is different from the connectivity of actual components. Furthermore, in EnergyPlus, a valve that adjusts a cooling coil's water flow rate is implicitly assumed to be part of the coil, which represents a component modularisation that does not correspond with that of actual systems. Further difficulties in generating a simulation model from a BIM can arise in non-conventional systems that do not follow the fluid loop structure of some building simulation programs. For example, because EnergyPlus (as well as DOE-2) does not allow equipment between thermal zones, the double facade system shown on the left side in Figure 6 had to be implemented using two air loops, and the exhaust fan had to be replaced by a supply fan to satisfy the EnergyPlus system structure. Automatically mapping the actual topology on the left from a BIM into the topology on the right seems to be an impossible task. The BIM translator would first need to understand in what system context the components are used, and then map one system topology, representing the actual building expressed in the BIM, into another system topology that can be simulated by the particular building simulation program. Because of this difficulty, early implementations of BIM to building simulation program translators are based on restricting the BIM instances to system topologies that can be simulated by the downstream simulation program (Bazjanac and Maile, 2004). In view of these limitations, more fundamental research

should be done to understand the computability of mappings between BIM and BIM-processing applications, as also suggested by Augenbroe et al. (2004).

To generate a mathematical model from a BIM, a more generic component modularisation and connectivity rule than what is employed in today's building simulation programs needs to be used. It is most natural to base the modularisation rule on how components are modularised in reality; typically modular units are defined by how they are packaged when shipped to the construction site. Furthermore, the connectivity rules of models should follow the same rules in which components can be connected in reality. This is easiest to realise by using acausal models that follow the object-oriented modeling paradigm and whose interface variables are sufficient to describe the interaction of connected objects. Models with these traits can be arbitrarily assembled to build systems as long as their connections have the same physical properties, such as temperature and heat flow rate for a solid-to-solid interface. Such a connection framework is indeed realised in the Modelica language.

A further difficulty arises when modeling cyber-physical systems. Cyber-physical systems will become more common as it becomes cheaper to densely instrument buildings with networks of sensor and actuators that control the building system. In cyber-physical systems, physical processes can affect when computations are performed and messages are sent across networks. For some systems, event detection, event ordering and other techniques for modeling and simulation of heterogeneous systems that properly model concurrency are needed. These needs may be easiest to meet by generating an actor-oriented system model for the communication network in which a model of computation controls the interactions between the different actors. The model may then be embedded into equation-based models for the physical components to form a hybrid system, which then can be used for designing and commissioning such networks.

Model translation and simulation

The translation of mathematical models into a simulation program can be fully automated. When translating a model, symbolic processing is important to reduce computing time since many building system simulation problems lead to large, sparse differential algebraic equation systems (DAE systems). Symbolic processing is typically used to reduce the index of the DAE system and to exploit sparsity. Pantelides (1988) presents an algorithm that reduces the index of DAE systems so that the equations can be solved using ordinary differential equation solvers. This algorithm is, for example, used by Dymola (Mattsson et al., 2000). It is current practice in modern simulators to exploit sparsity during model translation or during run-time. For example, Dymola and SPARK use symbolic processors during model translation to reduce the system of equations to block lower-triangular form. This process is called *partitioning* and is discussed in detail by Duff et al. (1989). Partitioning algorithms can guarantee that it is not possible to further reduce the dimensionality of the system of equations by permuting variables and equations. After partitioning, *tearing* can be used to break the dependency graph of equations and variables to further reduce the dimensionality of the system of equations. The resulting systems of equations are typically small but dense. Tearing can be done automatically or it can be guided by using physical insight with language constructs that can be embedded in a model library (Elmqvist and Otter, 1994). A further reduction in computation time can be obtained by symbolically inserting the discretisation formulae that represent the numerical integration algorithm into the differential-algebraic equation model, a process called *inline integration* that was introduced by Elmqvist et al. (1995). The IDA solver, in contrast, employs algorithms for sparse Jacobian factorisation

during simulation. It relies less on global symbolic manipulations during model translation, but it also employs tearing and automatic generation of Jacobians (Sahlin and Grozman, 2003). Fast model compilation is an advantage of IDA's approach that is particularly attractive for short-term simulations.
Research in symbolic and numerical solution algorithms and proper model formulation is critically important for making equation-based building modeling and simulation accessible to a large audience. It is not realistic to demand that the typical simulation user have knowledge in these mathematical fields. Therefore it is important that robust solution algorithms and model libraries be put in place so that they are available for use by typical simulation users without experience in fixing numerical problems. While these topics have received little attention from the building simulation community, the fact that IDA is used by HVAC engineers indicates that this problem is not insurmountable.

There are several technologies in place that decrease the computation time and lead to increased numerical robustness. But how do they compare to conventional building simulation programs? In general, comparing the run-time efficiency of different simulators is a challenging endeavour because of the different temporal and physical resolution of the tools, and because solvers can be tuned to be more efficient for certain classes of problems at the expense of robustness. Nevertheless, the following comparison shows that commercial general purpose simulators can be as efficient in performing a building simulation as commercial special tailored building simulation programs. Sahlin et al. (2004) compared the computation time of IDA ICE with EnergyPlus. In their numerical experiments, IDA ICE required approximately half the computation time that was required by EnergyPlus for a three zone building with natural ventilation and twice the computation time in initial experiments for a 53 zone building without natural ventilation. Sowell and Haves (2001) compared the computation time of SPARK and HVACSIM+ for a variable air volume flow system that serves six thermal zones. They report that SPARK computes about 15 to 20 times faster than HVACSIM+, and they attribute the decrease in computation times to SPARK's symbolic processing. Another variable air volume flow network model with 26 rooms shows that the computation time of SPARK and Dymola are similar for this class of problems (Wetter et al., 2008). Wetter and Haugstetter (2006) compared the simulation time of a multizone building model written in Modelica with a building model using TRNSYS' TYPE 56 that gives comparable energy use and transient response. The Modelica model was simulated using the solver of Dymola and Simulink. They report that Dymola computed four times slower and Simulink three times slower than TRNSYS. In these experiments, TRNSYS computed the heat conduction with Conduction Transfer Functions, which are considered to be computationally faster than the finite difference scheme used in the Modelica implementation.

One should keep in mind that these benchmarks have been simulated using a single processor. The recent switch to parallel microcomputers requires, however, that programs are written differently to take advantage of the new hardware architecture. For example, the conventional wisdom is that multiplications are slow but data loading and storing is fast. This is reversed by new hardware, because modern microprocessors can take 200 clock cycles to access dynamic random access memory (DRAM), but a floating-point multiplication may take only four clock cycles (Asanovic et al., 2006). Although how to best address this change in programming paradigm is an open research field, it is reasonable to expect that models formulated in equation-based, high-level programming languages, as opposed to existing simulation programs, are in a better position to take advantage of this shift towards parallel hardware. One reason for this expectation is that equation-based simulation environments naturally separate the algebraic and

numerical solvers from the model equations. This facilitates replacing algebraic and numerical solvers, as well as model translators, with newer versions that exploit the new hardware architecture.

Collaborative development

Over the past decade, Web 2.0 applications that enable distributed, concurrent, collaborative development of code and documentation have become a commodity. A characteristic of many successful open-source projects is that they employ a modular design that consists of small units with standardised interfaces. These units are good at doing one particular task, thereby providing a core of program functionalities around which further features can be built by a community of developers. Structuring the code into small units allows units to be developed and tested individually and then integrated into a larger system. It also reduces barriers to participation since only a small part of the program architecture needs to be understood by most contributors. This modularisation facilitates reusing code, distributing development efforts and using experts for the tasks that they are best at, such as developing numerical solvers, mathematical models or user interfaces. Involving different experts with a variety of backgrounds brings diverse points of view for solving a problem, which can further increase the quality of the code.

Many building simulation programs are developed by teams that do not always involve experts from other disciplines such as computer science, scientific computation and applied mathematics. Therefore, many advances in these fields are transferred to the building simulation community at a rather slow rate. To facilitate a better transfer of science and technologies, future developers of an environment for modeling and simulation should integrate, and where necessary, further develop existing technologies with other communities that are active in system modeling and simulation. This should enable sharing of models, solvers and interfaces to accelerate the development of more advanced tools. In such a model, significantly bigger investments for tool and library development can be leveraged.

A path towards a next generation environment for building simulation and computational engineering

Equation-based object-oriented modeling and simulation environments have been applied successfully in numerous demanding industrial applications. However, the building simulation community still rarely uses them, and little investment has been made to advance equation-based object-oriented building energy system modeling and simulation. If a new modular environment were to be used for whole building energy analysis, it would likely be measured against tools that have been developed for this purpose over several decades, often involving hundreds of man years of development time. Clearly, it does not make sense to target markets with heavy prior investment in existing technologies to help develop and mature a new environment for modeling and simulation. Instead, market segments with unmet needs should be pursued. Unmet needs are encountered primarily in the analysis and synthesis of algorithms for controls, fault detection and diagnostics, in research for building systems that integrate different domains such as HVAC, refrigeration, active facades and communication networks, and in the use of models during operation.

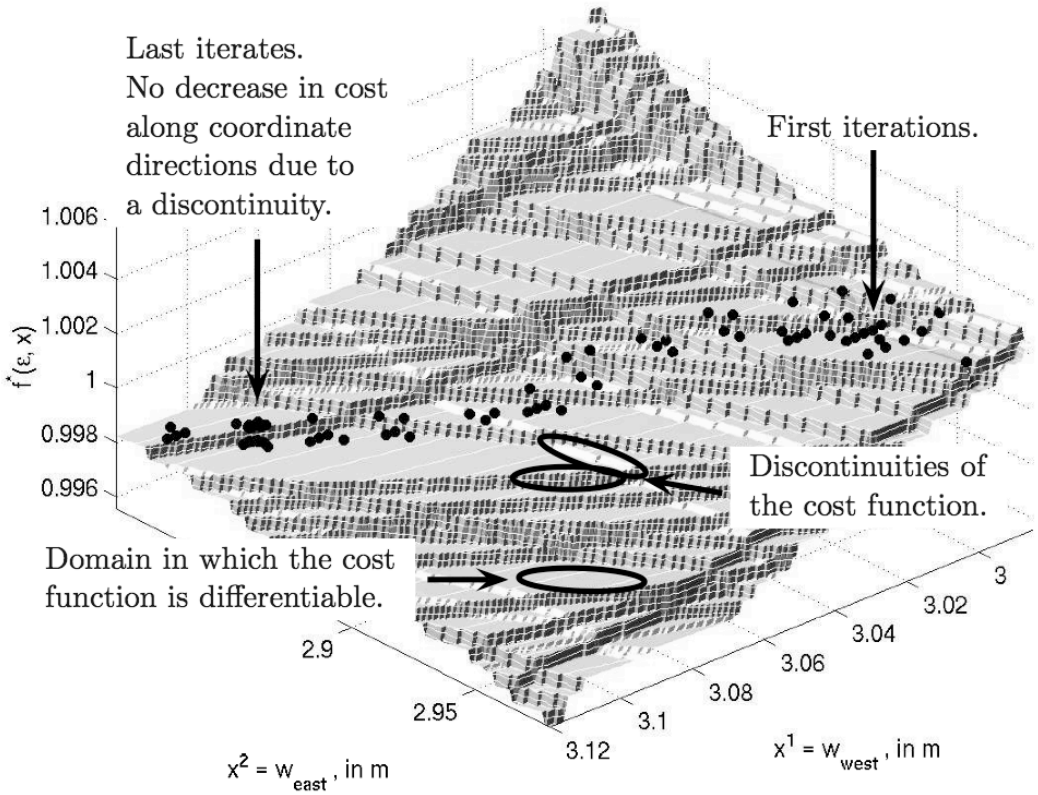


Figure 7: Parametric plot of the normalised source energy consumption for cooling and lighting as a function of the width of the west and east-facing window. The dots are iterates of the Hooke-Jeeves algorithm. (Figure adapted from Wetter and Polak (2004).)

Specific applications that a new modular environment for building modeling and simulation should target include:

- **Building systems for Net Zero Energy Buildings**. To achieve cost-effective Net Zero Energy Buildings, new system approaches are needed that require analysis and synthesis of coupled multi-physics systems that may include active facade, lighting controls, solar-assisted cooling and dehumidification, natural ventilation, storage of heat in the building structure and storage of humidity in desiccants. Such systems pose new challenges and opportunities for multi-variate control, optimal energy storage management and reduction of system-internal irreversibilities, i.e., destroyed exergy.
- **Design optimisation of innovative building energy and control systems.** Innovative building energy and control systems are typically harder to design since there are not many design rules available and because they often involve complex systems that interact dynamically. However, once simulation input files have been created, optimisation algorithms can help find the optimal building and energy system design with little user effort. In optimisation, a user selects a vector of design parameters x and defines a cost function $f(x)$ that needs to be minimised, possibly subject to constraints. Nonlinear programming algorithms, a class of computationally efficient optimisation algorithms, require the cost function to be differentiable in the design parameters. However, building simulation programs contain iterative solution algorithms and therefore compute a numerical approximation $f^*(\varepsilon, x)$ to the cost function, where ε is the tolerance of the

numerical solvers. The iterative solution algorithms introduce discontinuities in the numerical approximation to the cost function because a change in design parameters can cause a change in the number of solver iterations or in the grid on which a differential equation is discretised. Tightening the solver tolerance ε is possible in equation-based simulation programs, but it is often impractical in traditional building simulation programs since their solvers may be embedded in different component models and spread throughout hundreds of thousands of lines of code. Unfortunately, in building simulation programs, these discontinuities can be large enough to cause optimisation algorithms to fail far from an optimal solution (Wetter and Wright, 2004; Polak and Wetter, 2006). To illustrate this problem, we show in Figure 7 a section of a surface plot of normalised building energy consumption as a function of the width of the west and east windows, w_{west} and w_{east}. The plot was generated by sampling about 20,000 simulations on a two-dimensional grid. Globally, energy is minimised by increasing the width of both windows within the domain shown in the figure, i.e., the energy is minimal at the front corner in the figure. However, locally, due to the loose solver tolerance, the cost function exhibits "smooth" regions that are separated from each other by discontinuities. These "smooth" regions are visible in the graph as flat plateaus. The white lines that are visible on these plateaus are contours of equal function value. The problem is that in these "smooth regions," decreasing the west window width reduces energy. This caused the Hooke-Jeeves optimisation algorithm to generate a sequence of iterates that starts at the arrow on the right side in the figure and traverses towards the arrow in the left side in the figure where it jams at a discontinuity.

- **Model predictive control.** Model predictive control allows optimal equipment scheduling while taking into account the dynamics of the building energy system and the anticipated future building energy load. It allows one to minimise a user-specified cost function such as energy use, subject to state constraints like thermal comfort and indoor air quality. When solving the optimal control problem in a model predictive control algorithm, all state variables of the simulation model need to be reset to initial values prior to evaluating the performance of different control actions. However, in many traditional building simulation programs, resetting state variables is difficult if not impossible without significant code changes. Furthermore, for computational efficiency and to prove that the optimisation algorithm finds the optimal control sequence, the differential equations need to be smooth in the control input, which is typically not the case in building simulation programs. Lack of smoothness, in addition to the loose solver tolerance mentioned above, can cause the optimisation to fail for the same reasons as discussed in the previous item.
- **System linearisation for development of control algorithms.** Many design methods for control algorithms of nonlinear systems require linearisation of the open loop response with respect to controls or disturbances, sometimes at multiple operating points. Such linearisations are difficult to perform with large monolithic building simulation programs. The reason is that system linearisation requires isolating the subsystem that is part of the feedback control loop, initializing its state variables and simulating its open loop response subject to different control inputs and disturbances. An additional difficulty posed by large monolithic building simulation programs is that their numerical solvers frequently fail to converge. This can render a numerical approximation to a derivative useless for functions such as the one shown in Figure 7.
- **Fault detection and diagnostics.** For fault detection and diagnostics of components and subsystems, only a subset of the building system needs to be simulated. Realistic modeling of the component dynamics can be important to discern between faults and

transient effects. Such applications require modeling of realistic controls as opposed to the idealised controls that are found in many building simulation programs.

Theses applications do not in general require an annual whole building simulation. Rather they require a flexible modeling environment that allows a user to rapidly add new models, symbolically invert models and extract a subsystem of a model for further analysis, such as for input-output feedback linearisation or model reduction for controls design. By targeting these applications, the new environment can be developed with users who have the technical skills needed to mature the environment in terms of model scope, validation and formulation. This, in turn, will ensure robustness when obtaining a numerical solution and will ultimately form a base for a next generation building modeling and simulation environment.

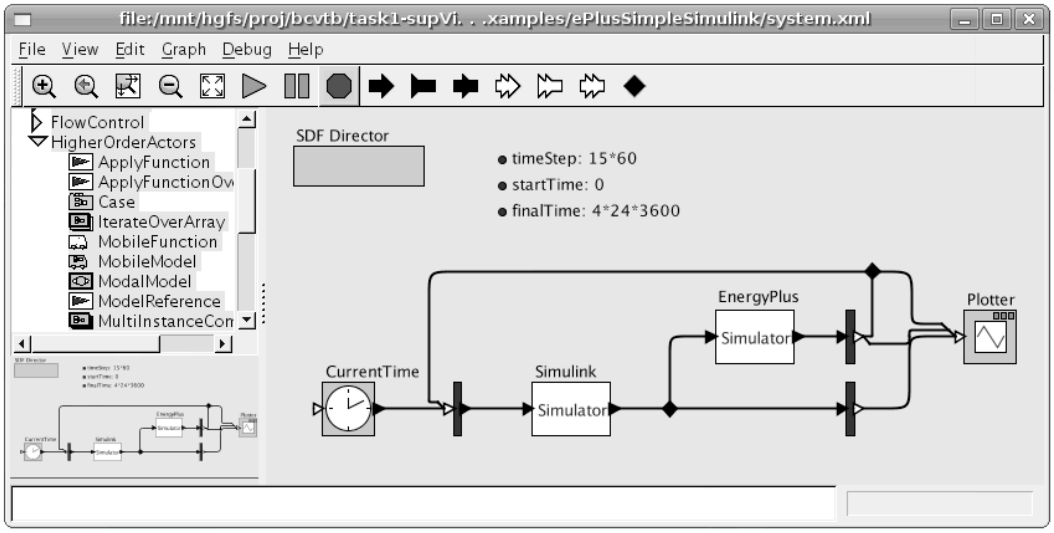


Figure 8: Ptolemy II graphical user interface with extensions to couple simulation programs, in this example EnergyPlus and MATLAB/Simulink, for run-time data exchange (Wetter and Haves, 2008).

While desirable for numerical efficiency it is often not necessary to model and simulate the whole building system in a new environment for modeling and simulation. Models expressed in equation-based object-oriented modeling languages can be interfaced with existing simulation programs to extend the scope of systems they can analyse. In environments that separate modeling from simulation, different models can be interfaced during compilation in such a way that they will share the same differential equation solver. An example is the commercial interface between Simulink and Dymola in which a Modelica model can be imported into Simulink in the form of an S-function. Monolithic simulation programs that do not allow extracting models from their solvers can be changed in such a way that they interface either directly another simulation program or a middleware that synchronises the data exchange as the simulation time progresses in each simulation program. A discussion about coupling of building simulation programs can be found in Trcka et al. (2006, 2009).
Both coupling approaches have been realised with many building simulation programs. As an example, Figure 8 shows the schematic editor of a modeling and simulation environment that has been extended to couple different building simulation programs (Wetter and Haves, 2008). Here,

each simulation program communicates during run-time using the Berkeley socket application programming interface to exchange data with an instance of a Java object that is shown in the figure by the white icons labelled “Simulator”. In this example, EnergyPlus simulates the building heat flow and natural ventilation and MATLAB/Simulink simulates the controller for the window opening. A model of computation controls the interactions between the models. In this modeling environment, different domains such as discrete event, continuous time and synchronous data flow can be combined hierarchically. This enables the simultaneous simulation of building energy systems, control systems and communication networks.

Synopsis

In a future environment for modeling and simulation, models will be expressed at a higher level of abstraction, which will allow a modeler to formulate the system closer to how the actual system behaves. Mathematical models and simulation programs will be distinct system representations; the first accommodates a human’s way of expressing and structuring systems and the second enables efficient computation of numerical solutions. Separating mathematical models from their solution processes will also facilitate assembling system models that combine different domains. Combining such models will be realised by formulating equality and conservation equations that link shared interface variables on an equation-level, as opposed to by integrating different programs, each with its individual solver, data management and input/output routines. Combining models on an equation-level is easier from the point of view of software engineering, mathematics (to ensure a consistent solution), and usability. Following the object-oriented modeling paradigm enables managing the complexity of large systems through hierarchical modeling. It also enables model assembly the same way as an experimenter would connect real components. This, in turn, allows creating a modeling environment where equation-based object-oriented models have the same modularity and connectivity rules as a modular BIM, facilitating the use of BIM to create simulation programs at different stages of the building life cycle. In addition, the higher level of abstraction allows sharing not only models, but also modeling and simulation environments and their associated technologies, such as numerical and symbolic solvers, with other industrial sectors.

Acknowledgements

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

K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, 2006, ‘The landscape of parallel computing research: A view from Berkeley‘, *Technical Report UCB/EECS-2006-183*, EECS Department, University of California, Berkeley.

G. Augenbroe, 2001, ‘Building simulation trends going into the new millenium’, in R. Lamberts, C. O. R. Negrão, and J. Hensen, (eds.), *Proc. of the 7-th IBPSA Conference*, vol. I, 15-27, Rio de

Janeiro, Brazil.

G. Augenbroe, P. de Wilde, H. J. Moon, and A. Malkawi, 2004, 'An interoperability workbench for design analysis integration', *Energy and Buildings*, 36 (8), 737-48.

P. A. Barnard, 2005, 'Software development principles applied to graphical model development', *Modeling and Simulation Technologies Conference and Exhibit*, American Institute of Aeronautics and Astronautics, San Francisco, CA.

V. Bazjanac and T. Maile, 2004, 'IFC HVAC interface to EnergyPlus - a case of expanded interoperability for energy simulation', *Proc. of SimBuild*, Boulder, CO.

V. Bazjanac, 2008, 'IFC BIM-Based Methodology for Semi-Automated Building Energy Performance Simulation', *Proc. of CIB-W78*, 25th Int. Conf. On Information Technology in Construction, Santiago de Chile, Chile.

F. Casella, M. Otter, K. Proelss, C. Richter, and H. Tummescheit, 2006, 'The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks', C. Kral and A. Haumer, (eds.), *Proc. of the 5-th International Modelica Conference*, vol. 2, 631-40, Vienna, Austria.

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

F. E. Cellier, H. Elmqvist, and M. Otter, 1995, 'Modeling from Physical Principles', in W.S. Levine (ed.) *The Control Handbook*, ISBN:0849385709, CRC Press, Boca Raton, FL.

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

J. A. Clarke, 2001, *Energy Simulation in Building Design,* Butterworth-Heinemann, Oxford, UK, 2nd ed.

I. S. Duff, A. M. Erisman, and J. K. Reid, 1989, '*Direct Methods for Sparse Matrices',* Monographs on Numerical Analysis, Oxford Science Publications, Oxford.

C. Eastman, K. Liston, R. Sacks, and P. Teicholz, 2008, *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors,* John Wiley & Sons.

H. Elmqvist, 1978, '*A structured model language for large continuous systems',* PhD thesis, Lund Institute of Technology, Lund, Sweden.

H. Elmqvist and M. Otter, 1994, 'Methods for tearing systems of equations in object-oriented modeling', in *European Simulation Multiconference*, 326-32, Barcelona, Spain.

H. Elmqvist, M. Otter, and F. Cellier, 1995, 'Inline integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems', *Proc. European Simulation*

Multiconference, xxiii-xxxiv, Prague, Czech Republic.

H. Elmqvist, H. Tummescheit, and M. Otter, 2003, 'Object-oriented modeling of thermo-fluid systems', in P. Fritzson (ed.) *Proc. of the 3rd Modelica conference*, 269-86, Linköping, Sweden.

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

P. Fritzson and V. Engelson, 1998, 'Modelica - A unified object-oriented language for system modeling and simulation', *Lecture Notes in Computer Science*, vol. 1445, 67-90, Springer, London, UK.

IBM Corporation, 1956, 'The Fortran Automatic Coding System for the IBM 704 EDPM', *IBM Applied Science Division and Programming Research Department*, New York, NY.

S. A. Klein and F. L. Alvarado, 1992, 'Engineering equation solver (EES)', *F-Chart Software*, Madison, WI.

K. P. Lam, N. H. Wong, A. Mahdavi, K. K. Chan, Z. Kang, and S. Gupta, 2004, 'SEMPER-II: an internet-based multi-domain building performance simulation environment for early design support', *Automation in Construction*, 13 (5), 651-63.

E. A. Lee, 2003, 'Model-driven development - from object-oriented design to actor-oriented design', *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation* (a.k.a. The Monterey Workshop), Monterey, CA.

E. A. Lee, 2006, 'Cyber-physical systems - are computing foundations adequate?'*, Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, Berkeley, CA

E. A. Lee and P. P. Varaiya, 2002, *Structure and Interpretation of Signals and Systems,* Addison Wesley.

A. M. Malkawi and G. Augenbroe, 2004, *Advanced Building Simulation*, 1st ed*,* Spon Press, New York, NY.

A. M. Malkawi, 2004, '*Immersive building simulation',* in A. M. Malkawi and G. Augenbroe (eds.), Advanced Building Simulation, 1st ed, Spon Press, New York, NY.

S. E. Mattsson and H. Elmqvist, 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*, Gent, Belgium.

S. E. Mattsson, H. Olsson, and H. Elmqvist, 2000, 'Dynamics selection of states in Dymola', *Modelica Workshop*, 61-67, Lund, Sweden.

C. C. Pantelides, 1988, 'The consistent initialization of differential-algebraic systems', *SIAM*

Journal on Scientific and Statistical Computing, 9 (2), 213-31.

E. Polak and M. Wetter, 2006, 'Precision control for generalized pattern search algorithms with adaptive precision function evaluations', *SIAM Journal on Optimization*, 16 (3), 650-69.

J. W. Polderman and J. C. Willems, 2007, *Introduction to Mathematical System Theory,* Texts in Applied Mathematics, vol. 26, Springer, New York, NY.

P. Sahlin, 1996, *Modeling and Simulation Methods for Modular Continuous Systems in Buildings,* PhD thesis, KTH, Stockholm, Sweden.

P. Sahlin, 2000, 'The methods of 2020 for building envelope and HVAC systems simulation - will the present tools survive?' *Dublin conference*, The Chartered Institution of Building Services Engineers, Dublin, Ireland.

P. Sahlin and P. Grozman, 2003, 'IDA simulation environment - a tool for Modelica based end-user application development', in P. Fritzson (ed.), *Proc. of the 3rd Modelica conference*, 105-14, Linköping, Sweden.

P. Sahlin and E. F. Sowell, 1989, 'A neutral format for building simulation models', *Proc. of the Second International IBPSA Conference*, 147-54, Vancouver, BC, Canada.

P. Sahlin, L. Eriksson, P. Grozman, H. Johnsson, A. Shapovalov, and M. Vuolle, 2004, 'Whole-building simulation with symbolic DAE equations and general purpose solvers', *Building and Environment*, 39 (8), 949-58.

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

E. F. Sowell and P. Haves, 2001, 'Efficient solution strategies for building energy system simulation', *Energy and Buildings*, 33 (4), 309-17.

E. F. Sowell, W. F. Buhl, A. E. Erdem, and F. C. Winkelmann, 1986, '*A prototype object-based system for HVAC simulation',* Technical Report LBL-22106, Lawrence Berkeley National Laboratory.

E. F. Sowell, W. F. Buhl, and J.-M. Nataf, 1989, 'Object-oriented programmingg, equation-based submodels, and system reduction in SPANK', *Proc. of the Second International IBPSA Conference*, 141-46, Vancouver, BC, Canada.

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

M. Trcka, J. L. M. Hensen, and A. J. T. M. Wijsman, 2006, 'Distributed building performance simulation - a novel approach to overcome legacy code limitations', *ASHRAE HVAC&R*, 12 (3a), 621-40.

M. Trcka, J. L. M. Hensen, and M. Wetter, 2009, 'Co-simulation of innovative integrated HVAC systems in buildings', *Journal of Building Performance Simulation,* 2(3), 209-230.

M. Vuolle, A. Bring, and P. Sahlin, 1999, 'An NMF based model library for building thermal simulation', in N. Nakahara, H. Yoshida, M. Udagawa, and J. Hensen (eds.), *Proc. of the 6-th IBPSA Conference*, Kyoto, Japan.

M. Wetter, 2005, 'BuildOpt - a new building energy simulation program that is built on smooth models', *Building and Environment*, 40 (8), 1085-92.

M. Wetter, 2006a, 'Multizone building model for thermal building simulation in Modelica', in C. Kral and A. Haumer (eds.), *Proc. of the 5-th International Modelica Conference*, vol. 2, 517-26, Vienna, Austria.

M. Wetter, 2006b, 'Multizone airflow model in Modelica', in C. Kral and A. Haumer (eds.), *Proc. of the 5-th International Modelica Conference*, vol. 2, 431-40, Vienna, Austria.

M. Wetter and C. Haugstetter, 2006, 'Modelica versus TRNSYS - A comparison between an equation-based and a procedural modeling language for building energy simulation', *Proc. of SimBuild*, Cambridge, MA.

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

M. Wetter and E. Polak, 2004, 'A convergent optimization method using pattern search algorithms with adaptive precision simulation', *Building Services Engineering Research and Technology*, 25 (4), 327-38.

M. Wetter and J. Wright, 2004, 'A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization', *Building and Environment*, 39 (8), 989-99.

M. Wetter, P. Haves, M. A. Moshier, and E. F. Sowell, 2008, 'Using SPARK as a solver for Modelica', *Proc. of SimBuild*, Berkeley, CA.

Recommended reading

For a computational approach to heterogeneous systems the reader is referred to the textbook of Lee and Varaiya (2002). For a discussion about trends and challenges of system level design we refer to Sangiovanni-Vincentelli (2007). The prerequisites for generating efficient simulation code from a general-purpose solver are discussed by Elmqvist et al. (1995). Cellier et al. (1996) discusses common pitfalls and misconceptions about the modeling of physical systems. Tiller (2001) and Fritzson (2004) discuss different equation-based modeling approaches for dynamical systems. Cellier and Kofman (2006) present in a rigorous way methods for transforming mathematical models into computational code. The textbook has been written for an engineering audience. Mattsson et al. (2000) describe how symbolic and numerical methods are combined in Dymola to solve reliably and efficiently high-index DAE systems. For a detailed discussion of

BIM see for example Eastman et al. (2008). Clarke (2001) gives a detailed exposition of models that are typically found in building simulation programs. Malkawi and Augenbroe (2004) present recent trends in building simulation with a focus on combined air and heat flow, uncertainty propagation and integration of simulation into the design process.

Assignments

1) Implement two versions of the system model shown in Figure 3(a). For the first version, use causal block-diagrams and for the second version, use acausal models for the physical components. As parameters, use for each heat capacitor *10 J/K*, for each heat conductor *10 W/K*, for the convective heat transfer *5 W/K*, for the setpoint and the initial temperatures *20°C* and for the control gain *100 W/K*. For the boundary condition, use $T(t) = 20 + 5 \sin(2 \pi t/3600)$, where t denotes time in seconds and the temperature is in degree Celsius. Compute the maximum control error. Next, reconfigure the system to the topology shown in Figure 3(b). Compute again the maximum control error. Discuss the time it took to develop and reconfigure the two model representations.
Hint: You may use Modelica or MATLAB/Simulink with the SimScape™ extension to create acausal and causal models.

2) How do you think a next generation building energy analysis tool will look like?

Answers

1) For configuration (a), the maximum control error is *0.12 K* and for configuration (b), it is *0.24 K*.