Proceedings of BS2013:
13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26-28

USING GENERAL MODELING CONVENTIONS FOR THE SHARED DEVELOPMENT OF BUILDING PERFORMANCE SIMULATION SOFTWARE

Rhys Goldstein, Simon Breslav, and Azam Khan
Autodesk Research, 210 King St. East, Toronto, ON, Canada

ABSTRACT

The building performance simulation community applies theory from several different fields to develop models for heat transfer, light propagation, human behavior, and other domains. To integrate these models, we propose the adoption of general modeling conventions from the less familiar field of modeling and simulation theory. The conventions we explore are known as the Discrete Event System Specification (DEVS). With DEVS, a model-independent simulator responsible for advancing time alleviates many of the technological difficulties involved in coupling models. We show how DEVS, from a mathematical perspective, accommodates the co-simulation strategies known as loose and strong coupling as well as strategies involving variable time steps. We also show how a model based on a functional decomposition of a system, as opposed to a topological decomposition, readily supports the sharing of domain-specific algorithms. The examples presented were implemented using DesignDEVS, an environment we created to help communities of researchers collaborate in the development of simulation software.

INTRODUCTION

The ongoing pursuit of increasingly accurate and efficient building performance simulation (BPS) software, addressing an expanding array of design applications, necessitates the sharing and integration of simulation code developed by experts in different domains. This need for shared development is widely acknowledged and follows from two observations. First, inventing a state-of-the-art model for any individual domain, be it heat transfer, light propagation, mechanical equipment, control logic, the outdoor environment, or human behavior, requires considerable dedication on the part of a single research team. Second, simulation-based building design applications must account for multiple domains. The task of coupling single-domain models of different research teams is complicated by the many interactions between domains that occur over time (Hensen, 2002).

While the BPS community applies theory from different fields to develop single-domain models, a large body of research relevant to the coupling of these models has barely been explored. Modeling and simulation theory includes sets of general modeling conventions that strive to facilitate representations of any type of real-world system. Some of these conventions, such as Bond Graphs, Statecharts, and Petri Nets, rely on diagrams. Others use mathematical notation, including the Discrete Time System Specification (DTSS), the Differential Equation System Specification (DESS), and the Discrete Event System Specification (DEVS), as defined in *Theory of Modeling and Simulation* (Zeigler et al., 2000).

Of these conventions, DEVS seems most likely to support the diverse interests of a research community. It separates the simulation process from the models that describe system behavior, and facilitates coupling in a manner that avoids explicit dependencies between interacting models. In this paper, we explore the application of DEVS to the shared development of BPS software from three perspectives. From a technological perspective, we compare the use of a DEVS-based simulator to that of model-dependent simulators such as ESP-r and EnergyPlus, other model-independent simulators such as Modelica and Ptolemy II, and co-simulation. From a mathematical perspective, we demonstrate how DEVS accommodates the co-simulation strategies known as loose and strong coupling as well as strategies involving variable time steps. Finally, from a modeling perspective, we contrast the use of topological and functional system decompositions for structuring DEVS models involving multiple building domains.

TECHNOLOGICAL PERSPECTIVE

Most simulation software is based on one of three approaches:

- Model-Dependent Simulator
- Model-Independent Simulator
- Co-simulation

A model-dependent simulator is a simulation program in which the code fulfilling the role of a simulator, the advancement of time and the initiation of the different phases of a simulation, is integrated with the code fulfilling the role of a model, the representation of a real-world process or system. The development of large model-dependent simulators, such as the BPS tools ESP-r and EnergyPlus, can be shared between programmers with access to the same code repository. This is how most software systems are developed, including applications for business and entertainment. However, this form of collaboration often requires contributors to have a common vision for fu-

ture versions of the software. A common vision may be difficult to attain with different research teams seeking to explore different combinations of simulation algorithms. In practice, research teams tend to preserve their autonomy by maintaining their own model-dependent simulators. As a consequence, a considerable amount of work is needed to extend one simulator with code developed for another.

A model-independent simulator is a program that initiates the different phases of a simulation and advances simulated time, but does not explicitly reference any code representing a real-world process or system. Representing the real world is left to the model developers who use the simulator. A model-independent simulator can be reused with any model implemented according to the simulator's modeling conventions. The commercial product Dymola and the open source environment OpenModelica are examples of model-independent simulators which use the Modelica language as the set of modeling conventions. With the recent introduction of a library of models for HVAC components and related processes (Wetter, 2009), Modelica has received considerable attention as an option for BPS. Other model-independent simulators relevant to the BPS community include Ptolemy II and the TRNSYS kernel.

Co-simulation, in which multiple simulators run simultaneously while exchanging information, is sometimes the most tractable way to combine models of interacting systems that have already been developed in separate programs. Co-simulation can be implemented by modifying one simulator to control the others, as done by Janak (1997) to have ESP-r influence and access the results of the lighting simulation tool Radiance. Alternatively, a new program can be written to control all the simulators. Beausoleil-Morrison et al. (2011) follow this approach for the coupling of ESP-r with TRNSYS, a package that includes the model-independent TRNSYS kernel and a library of models for renewable and other energy systems. The Building Controls Virtual Test Bed (BCVTB) uses Ptolemy II to connect EnergyPlus, Radiance, Modelica, and other simulators and software tools (Wetter, 2011). The use of a consistent communication mechanism makes the BCVTB's approach to co-simulation compelling from a technological point of view. However, it introduces mathematical limitations that we will discuss later.

The advantage of model-independent simulators is that they alleviate many of the technological difficulties involved in coupling models. For model-dependent simulators, the difficulty lies in breaking the dependence of various source files or object-oriented classes on one simulation program, and merging the extracted code with another simulation program. Co-simulation frequently involves modifying both simulators to send and receive information, controlling one or more simulators from another program, and writing low-level, inter-process communication code. Connecting two models written for a model-independent simulator, by contrast, can be as simple as drawing an arrow in a graphical editor. Of course it is never that easy due to mathematical and modeling considerations, which we will discuss later. But from a strictly technological perspective, a model-independent simulator can dramatically simplify the task of integrating multiple teams' developments.

The drawback to model-independent simulators is that they require model developers to embrace a common set of modeling conventions. This is particularly challenging for conventions perceived as unfamiliar, unnecessarily complex, or overly restrictive. With the Modelica language, lack of familiarity is a key issue. Its use of equation-based modeling, in which differential equations are encoded directly and solving code is generated automatically, is a significant departure from traditional imperative programming. Complexity is a concern with Ptolemy II, as its models may be based on communicating sequential processes, continuous time, discrete events, process networks, or synchronous dataflow (Eker et al., 2003). The conventions of the TRNSYS kernel are less diverse but arguably more restrictive. They do not support variable time steps, for instance.

There are several reasons to think that the modeling conventions known as DEVS could be embraced by different research teams within a community. First, DEVS models are almost always implemented using the familiar imperative style of programming involving assignments, if statements, and loops. Second, DEVS is a relatively minimalistic set of conventions. It consists of atomic models, which when implemented contain most of the imperative code, and coupled models, which connect any number of other models of either type. Third, since its first introduction in 1976, DEVS has been known for its generality. The fact that an atomic model's transitions functions depend not only on its state, but also on the time elapsed since the previous transition, allows DEVS to represent essentially any time-varying system.

Let us introduce DEVS with an example. The equations below represent a single-zone building with an indoor air temperature of T. The indoor temperature is influenced by the outdoor temperature T_{out}, and by heat transferred from an idealized HVAC system at a rate of $\dot{q}_{sys}$.

$$T_{out}(t) = \bar{T}_{out} - \Delta T_{out} \cdot \sin\left(\frac{2 \cdot \pi \cdot t}{d}\right)$$

$$\dot{T}(t) = \frac{\sum_i U_i \cdot A_i}{C} \cdot (T_{out}(t) - T(t)) + \frac{\dot{q}_{sys}(t)}{C}$$

$$\dot{q}_{sys}(t) = UA_{sys} \cdot (T_{set} - T(t))$$

The details of this mathematical model are as follows. The outdoor temperature is a sinusoidal function of time with a fixed initial and mean value of $\bar{T}_{out}$, a

fixed amplitude of ΔT_{out}, and a period of d which we assume to be one day. The equation for the indoor temperature is a standard albeit simplistic model: the indoor air has a heat capacity C, the product of air density ρ (1.2 kg/m^3), specific heat capacity c_p (1000 J/(kg·K)), and volume V; each surface i bounding the indoor air has a U-value U_i and an area A_i. The rate of heat transfer from the HVAC system is proportional to the difference between the indoor temperature and a fixed setpoint temperature T_{set}. The positive constant UA_{sys} determines how aggressively the system strives to maintain the setpoint.

Because T is the only variable that is differentiated, the three equations combined yield a single ordinary differential equation. The problem was contrived to permit an analytic solution, useful for verifying the numerical solution implemented with DEVS. For the sake of convenience, we define four constants.

$$r_{env} = \frac{\sum_i U_i \cdot A_i}{C} \qquad r_{sys} = \frac{UA_{sys}}{C}$$

$$r_{tot} = r_{env} + r_{sys} \qquad r_{day} = \frac{2 \cdot \pi}{d}$$

Below is the analytic solution.

$$T(t) = \left(T_0 - \frac{r_{env} \cdot r_{day}}{{r_{tot}}^2 + {r_{day}}^2} \cdot \Delta T_{out} \right) \cdot e^{-r_{tot} \cdot t} + \frac{r_{env} \cdot \bar{T}_{out} + r_{sys} \cdot T_{set}}{r_{tot}} \cdot \left(1 - e^{-r_{tot} \cdot t}\right) + \frac{r_{env} \cdot \cos\left(r_{day} \cdot t + \tan^{-1}\left(\frac{r_{tot}}{r_{day}}\right)\right)}{\sqrt{{r_{tot}}^2 + {r_{day}}^2}} \cdot \Delta T_{out}$$

For the numerical solution, each of the three original equations is treated in its own atomic DEVS model: a Weather model for the T_{out} equation, a Thermal model for T and $\dot{T}$, and an HVAC model for $\dot{q}_{sys}$. A single coupled DEVS model, illustrated in Figure 1, defines links through which the three submodels interact. Note that the Weather and Thermal models have a one-way relationship, as the outdoor temperature influences the indoor temperature. The Thermal and HVAC models are connected to form a feedback loop, reflecting the fact that the indoor temperature and the system heat transfer rate influence one another. From a technological perspective, coupling DEVS models requires little more than the creation of such diagrams in a graphical editor.

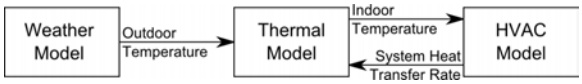

Figure 1: A coupled DEVS model.

An atomic model consists primarily of an external transition function and/or an internal transition function. The transition functions of the Weather, Thermal, and HVAC models are specified mathematically in Figure 2.

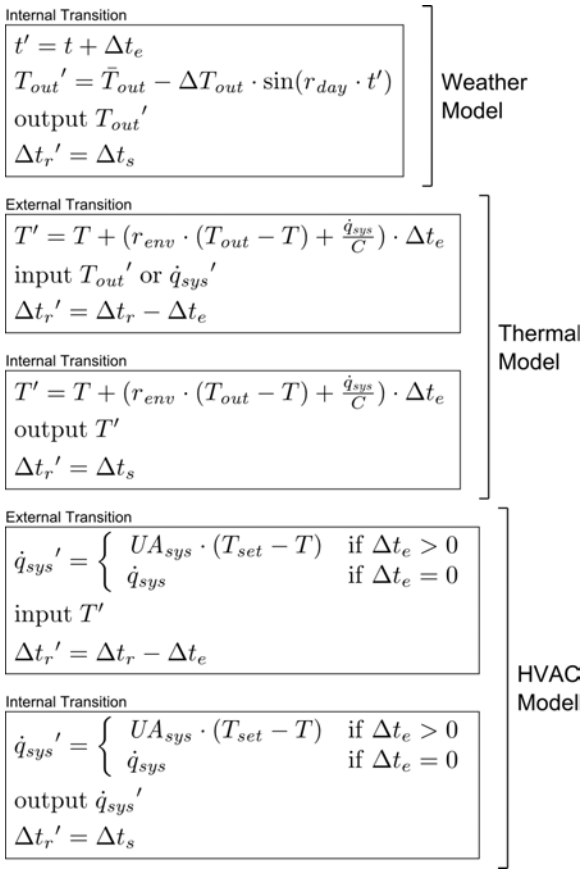

Figure 2: Three atomic DEVS models.

A transition is an event at which a model may undergo a state change, and may interact with other models via inputs and outputs. The purpose of an external transition is to receive the input that triggers it. As indicated in both Figure 1 and the external transition functions of Figure 2, the Thermal model can receive either of two inputs while the HVAC model receives one. The Weather model receives no inputs, and therefore lacks an external transition function. An internal transition is scheduled by the model itself, and has the option of producing outputs. The internal transition functions in this example indicate that all three atomic models output a single quantity, which is consistent with Figure 1. Note that more complex inputs and outputs are possible. To achieve a more realistic BPS simulation, the Weather model could output an array of weather variables, and the Thermal model could output a vector of temperature values describing each of a building's thermal masses.

Transition functions deal with two important duration variables: the read-only time Δt_e elapsed since the previous transition, and the modifiable time Δt_r remaining until the next internal transition. An internal transition occurs only if Δt_r elapses before any input is received. If an input is received, an external transitions occurs immediately and the next internal transition is rescheduled according to the updated remaining time ${\Delta t_r}'$. To give a more complete description of DEVS, we should mention that the remaining time is traditionally supplied not by a variable, but by a time

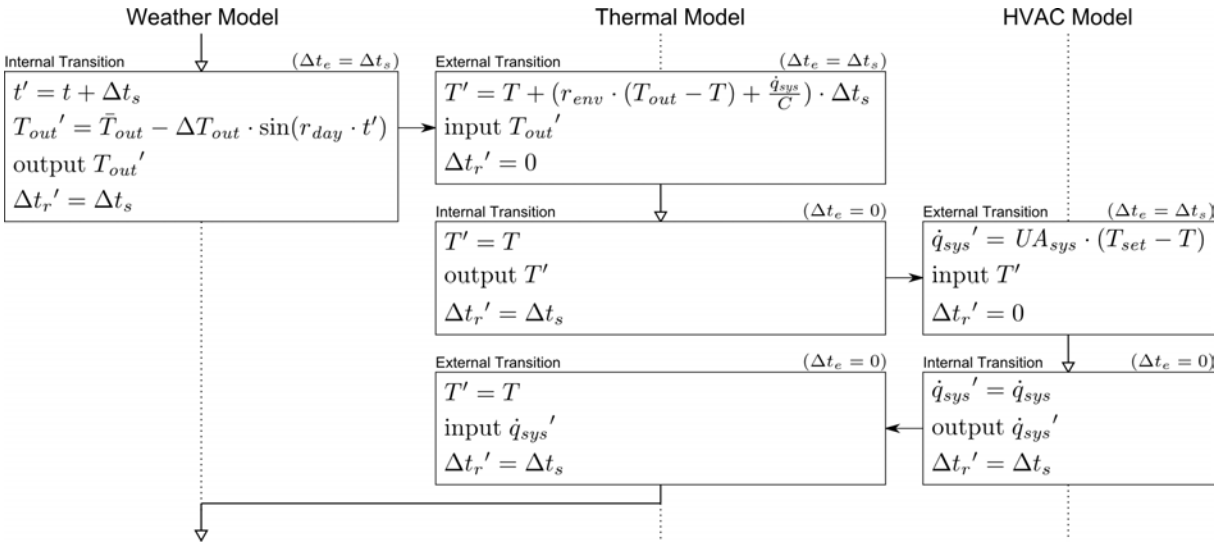

Figure 3: Sequence of external and internal transitions occurring at a single time step.

advance function invoked after every transition. In this paper we assume that the time advance function always yields the state variable Δt_r, and therefore we omit this function.

There are various ways for a DEVS model to perform calculations. The Weather model keeps track of the absolute time by adding the elapsed time Δt_e to a state variable t at the beginning of every transition. This time value is used to calculate T_{out}, which is then output. The Thermal and HVAC models are complicated by the fact that an advancement of time can be interrupted by either an external transition or an internal transition. Thus they are prepared to perform the same calculation, either an extrapolation of T or a direct evaluation of $\dot{q}_{sys}$, in both transition functions.

DEVS does not require a fixed time step, but does allow it. In this example the time step Δt_s is common to all three atomic models. Each internal transition schedules the next internal transition after a duration of Δt_s. The two external transitions are more complicated: if a time step has elapsed ($\Delta t_e = \Delta t_s$), they must schedule an immediate output ($\Delta t_r{}' = 0$); but if no time has elapsed ($\Delta t_e = 0$), they must leave the remaining time unchanged ($\Delta t_r{}' = \Delta t_r$). The assignment $\Delta t_r{}' = \Delta t_r - \Delta t_e$, which is typical of DEVS models in general, covers both cases.

The last DEVS concept illustrated by this example is the ordering of simultaneous events. There are two rules. First, if a model receives an input at the exact time it is scheduled to undergo an internal transition, the input takes precedence. Second, if multiple models are scheduled to undergo internal transitions at the same time, an ordering determines which occurs first. Figure 3 gives the sequence of transitions that occur at each time step assuming that the Weather model is given the highest priority, followed by the Thermal model, followed by the HVAC model. At the beginning of a time step, all three models are scheduled to undergo an internal transition. Having the highest priority, the Weather model goes first. Its internal transition produces an output that is received by the Thermal model, triggering an external transition. According to the first rule, this external transition preempts the Thermal model's internal transition. Applying the same rules yields the remainder of the sequence.

Note that there are six possible orderings of the three atomic models. With a different ordering, not only would the transitions occur in a different sequence, but either of the calculations shown in the external transitions might be performed instead in an internal transition. Nevertheless, for all six orderings, the output of each atomic model remains the same. This desirable property was achieved not by analyzing all six cases, but by defining the atomic models according to constraints that guarantee robustness regardless of how models are ordered. In this case, the atomic models recalculate temperatures and heat transfer rates only after an advancement of time, and output these variables exactly once per time step. Other strategies involve different constraints. In the next section, we will solve the same equations with DEVS models capable of producing multiple outputs per time step.

MATHEMATICAL PERSPECTIVE

DEVS offers research communities the possibility of integrating simulation code while avoiding many of the technological difficulties associated with sharing access to interdependent source files or establishing communication protocols between multiple simulators. From a strictly technological point of view, coupling DEVS models requires little more work than drawing arrows in a graphical editor. Of course it is never that easy, as there are mathematical and modeling considerations that transcend computing technology. From a mathematical perspective, it is necessary that all interacting models are compatible in terms of the strategies used to solve systems of equations in a

co-operative fashion. We consider four such strategies:

- Definitive (discrete time, continuous state)
- Speculative (discrete time, continuous state)
- Adaptive (continuous time, continuous state)
- Quantized State (continuous time, discrete state)

DEVS accommodates all four strategies, which is appropriate for a community pursuing multiple options. The BCVTB, possibly the BPS community's most extensible co-simulation effort, is limited to the definitive strategy in which each part of the current state of a system is updated once per time step. The ESP-r/TRNSYS co-simulator of Beausoleil-Morrison et al. (2011) is a recent example of the speculative strategy, where parts of the future state are estimated then repeatedly revised within each time step. The definitive and speculative strategies are more commonly referred to as *loose coupling* and *strong coupling* (Trčka et al., 2009), emphasizing differences in how models are connected in co-simulation projects. We adopt new terminology because, with DEVS, the coupling of models varies little if at all between strategies.

The definitive and speculative strategies are similar to one another in that they discretize time into fixed steps while allowing state variables such as zone temperatures to remain continuous. The lesser explored adaptive and quantized state strategies, by contrast, are differentiated by their representation of time and state. Both strategies treat time as continuous by permitting variable time steps. With the adaptive strategy, time steps are shortened to improve accuracy over time periods of rapid change, but lengthened to reduce computational requirements over periods of gradual change. It has been applied to BPS by Zimmermann (2001) and Gunay et al. (2013), the latter using DEVS. The quantized state strategy is similar, except that time steps are varied in a way that constrains each state variable to a set of discrete values (Cellier and Kofman, 2006).

Let us revisit the example with the atomic models named Weather, Thermal, and HVAC. Whereas the definitive strategy was used in the previous section, here we will solve the same equations using the speculative strategy. The coupled model of Figure 1 remains unchanged, but all three atomic models must be redefined to output not only the current value of the outdoor temperature, indoor temperature, or system heat transfer rate, but also the predicted value for the next time step. For the Weather model the change is trivial, since we can evaluate the outdoor temperature at any requested time. For the HVAC model, matters are complicated by the need for external iterations, which we will explain shortly. It is the Thermal model, however, that best illustrates the speculative strategy, as the new version involves both external and internal iterations within each time step. We will present an implementation of this model.

The examples in this paper were implemented using DesignDEVS, an environment we created to help communities of researchers collaborate in the development of simulation software. Using DesignDEVS, atomic DEVS models are written in the general-purpose Lua programming language. Below is the Lua code for the speculative Thermal model's external and internal transition functions.

```lua
if elapsed() == dts then
    T[1] = T[2]
    Tout[1] = Tout[2]
    q_sys[1] = q_sys[2]
    k = 0
elseif elapsed() ~= duration(0) then
    error("inconsistent time step encountered")
end

local port, value = input()
if port == "Outdoor Temperature" then
    Tout = copy(value)
elseif port == "System Heat Transfer Rate" then
    q_sys = copy(value)
end

dtr = duration(0)
```

```lua
if elapsed() == dts then
    T[1] = T[2]
    Tout[1] = Tout[2]
    q_sys[1] = q_sys[2]
    k = 0
end

if k < k_max then
    local dt = dts/duration(1, "seconds")
    local Tout_mean = (Tout[1] + Tout[2])/2
    local q_sys_mean = (q_sys[1] + q_sys[2])/2
    local Tnext = T[2]
    local j = 0
    local dTnext = math.huge
    while j < j_max and dTnext > dTeps do
        local Tmean = (T[1] + Tnext)/2
        local Tprev = Tnext
        Tnext = T[1] + (r_env*(Tout_mean - Tmean)
                        + q_sys_mean/C)*dt
        Tnext = (Tprev + Tnext)/2
        dTnext = math.abs(Tnext - Tprev)
        j = j + 1
    end
    Tnext = (T[2] + Tnext)/2
    local dT = math.abs(Tnext - T[2])
    if k == 0 or dT > dTeps then
        T[2] = Tnext
        output("Indoor Temperature", T)
        k = k + 1
    end
end

dtr = dts
```

Listing 1: External (top) and internal (bottom) transition code for the speculative Thermal model.

The indoor temperature `T`, outdoor temperature `Tout`, and system heat transfer rate `q_sys` are 2-element vectors storing both current and future values. As with the definitive Thermal model, the current values (`T[1]`, `Tout[1]`, `q_sys[1]`) are updated in both transition functions, but only if time has elapsed (`elapsed() == dts`, where `dts` is the time step). The current values are updated using the future values (`T[2]`, `Tout[2]`, `q_sys[2]`), which have already been calculated.

Let us now identify the external and internal iterations in this example. The internal iterations seek a potential future indoor temperature `Tnext` consistent with all available current and future variables. They occur

entirely within the Thermal model's internal transition in the `while` loop at line 15. External iterations seek mutually consistent future values for both the Thermal model (`T[2]`) and the HVAC model (`q_sys[2]`). Because the two models must exchange information, each external iteration requires a pass through both transition functions. In each pass, a decision is made at line 26 in the Thermal model's internal transition function. If it is the first pass, or if the updated `Tnext` differs significantly from the previous prediction, then `Tnext` is recorded as `T[2]` and output. Receiving this future temperature, the HVAC model might respond with a new `q_sys[2]`. The Thermal model would then undergo an external transition, record the input (`q_sys = copy(value)`), schedule an immediate internal transition (`dtr = duration(0)`, where `dtr` is the remaining time), and eventually proceed to line 26. If on line 26 the new temperature prediction is sufficiently similar to the previous prediction, the cycle of outputs is broken and time advances.

Unlike strongly coupled simulators, speculative DEVS models require no custom external module to coordinate them. The models are independently responsible for enforcing iteration limits (e.g. `j_max`, `k_max`), and applying relaxation as described in Trčka et al. (2009) (e.g. lines 20 and 24).

Figure 4 shows simulation results produced by the definite and speculative examples. Indoor temperature profiles obtained with 2-hour and 30-minute time steps are plotted alongside the analytic solution given in the previous section.

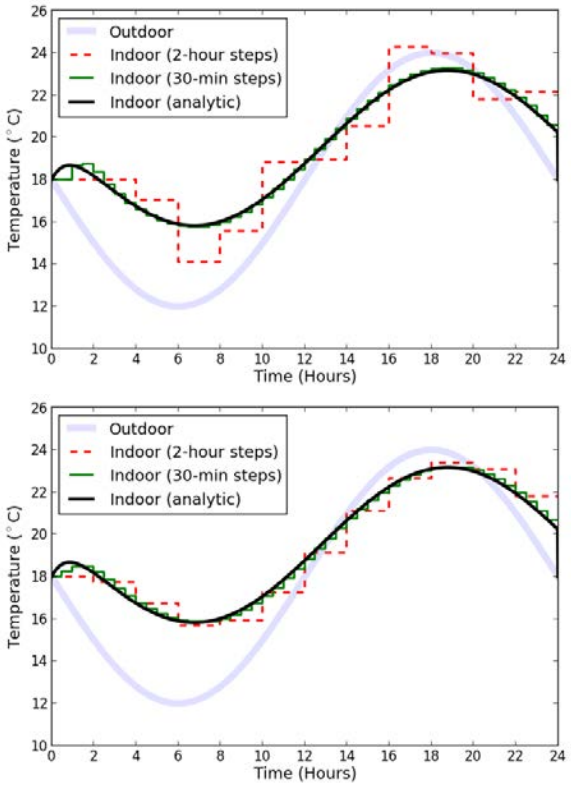

Figure 4: Definitive (top) and speculative (bottom) results for the Weather/Thermal/HVAC simulations.

The simulated building has a $12\ \mathrm{m} \times 9\ \mathrm{m}$ footprint, a height of $4\ \mathrm{m}$, and wall and roof U-values of 0.5 and $0.3\ \mathrm{W/(m^2{\cdot}K)}$ respectively. The outdoor temperature profile has a mean value of $18°\mathrm{C}$ and an amplitude of $6°\mathrm{C}$. For the HVAC system, the setpoint temperature is $22°\mathrm{C}$, and $UA_{sys} = 70\ \mathrm{W/K}$. The initial indoor temperature is $18°\mathrm{C}$.

With 2-hour time steps, the definitive results show signs of instability while the speculative results agree reasonably well with the analytic solution. We do not conclude that the speculative strategy is superior, however, since the definitive strategy will allow considerably shorter time steps given the same amount of time and computing power. The relative performance of the two strategies is studied in Trčka et al. (2007). Our purpose is simply to demonstrate that DEVS accommodates both. With 30-minute time steps, both sets of results agree well with the analytic solution.

The examples presented thus far demonstrate how mathematical strategies developed for co-simulation can be applied using a single DEVS-based simulator. It is equally important to understand that, by allowing the remaining time to be calculated at every transition, DEVS supports the lesser explored strategies based on variable time steps. Our focus now shifts from mathematics to modeling, but note that the example in the next section employs the quantized state strategy.

MODELING PERSPECTIVE

To fully benefit from DEVS, or any modular approach to simulation development, it not enough to master the mechanics of composing and combining models. One must also understand the implications of various modeling decisions, which become increasingly important as simulation models grow in complexity. A key decision is the type of system decomposition that collaborating research teams should agree to use as a basis for structuring their coupled models:

- Topological Decomposition
- Functional Decomposition

Figure 5 illustrates the two approaches. With a topological decomposition, each submodel is associated with a spatial element such as a wall segment, a zone, or an occupant. Accordingly, links between components of a coupled model reflect the spatial topology of a real-world system. A functional decomposition, by contrast, associates each submodel with an aspect of the entire system such as the thermal domain or the occupancy domain of a building. Links exist wherever there are interactions between domains. Model-dependent simulators and co-simulation projects tend to favor functional decompositions, as most simulation algorithms are domain-specific. Model-independent simulators typically support both types of decompositions, though the topological approach is often emphasized. Describing the topology-based modeling of HVAC systems using Modelica, for example, Wetter (2009) explains that having each model compo-

nent represent a physical device is a tenet of the language. The thermal domain has also been tackled with a model-independent simulator in conjunction with a topological decomposition (Zimmermann, 2001). This method essentially replaces the matrix-based algorithms described in Clarke (2001) and elsewhere with the rapid exchange of temperature and heat flow values between adjacent building elements.

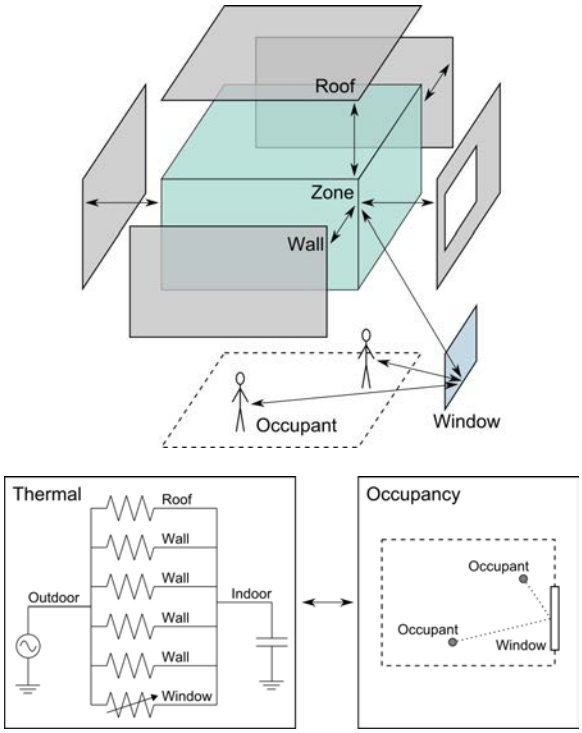

Figure 5: Illustration of models based on topological (top) and functional (bottom) system decompositions.

Although topological decompositions can be appealing when depicted for simple examples, complications may arise for models sufficiently detailed to capture the many performance-influencing elements of a real building. One complication is the need to generate coupled models with hundreds of interconnected components. Graphical editors, while adequate for small models, are rarely useful for authoring such complex networks. A related issue is the computational overhead associated with the interactions between these numerous components. A third problem with topological decompositions is that some mechanism other than model coupling may be needed to support collaboration. To understand why, consider that each submodel of a topology-based network represents a building element relevant in multiple domains. If research teams each specialize in a particular domain, they will likely end up with overlapping sets of submodels that cannot be integrated by coupling alone.

Techniques addressing the challenges associated with topology-based models have been presented and are likely to mature over time. However, to avoid these complications in the short term, we advise prospective DEVS users in the BPS community to consider functional decompositions of buildings and their surrounding environments. Even relatively detailed coupled models based on functional decompositions are likely to include less than two dozen or so components, a small enough number that graphical editors remain useful and computational efficiency need not be severely compromised. Also, having worked independently on separate domain-specific algorithms, research teams can use model coupling as the primary mechanism to integrate their developments.

To illustrate the use of DEVS with a functional decomposition of a building, consider a simple DEVS model based conceptually on Figure 5 and structured according to Figure 6. As with the example treated in preceding sections, the Weather model supplies the outdoor temperature to a single-zone Thermal model. However, one of the walls bounding the zone now has a window that can be opened and closed by the building's occupants, affecting the rate at which the indoor temperature changes. Both indoor and outdoor temperatures influence occupants' decisions on whether to manipulate the window.

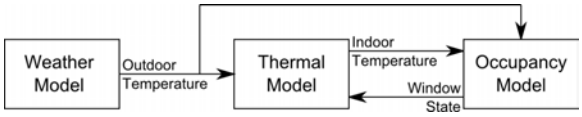

Figure 6: Coupling of functional models.

Let us briefly explain the mathematics of this example before we return to the subject of decomposition. The Thermal model solves the same differential equation as in the earlier example, except that $\dot{q}_{sys} = 0$ and the outdoor temperature T_{out} is assumed to remain constant between inputs received from the Weather model. If T is the indoor temperature at the preceding transition, and a duration of Δt_e has elapsed since that transition, then the new indoor temperature T' can be calculated as follows.

$$T' = T_{out} + (T - T_{out}) \cdot e^{-\frac{\sum_i U_i \cdot A_i}{C} \cdot \Delta t_e}$$

This version of the Thermal model uses the quantized state strategy, and for that reason the right-hand side above is evaluated only when an input is received. If instead the temperature reaches the threshold T_{next}, which is any multiple of a fixed temperature interval, then T' is simply T_{next} and this value is output. To determine when such an internal transition should occur, the remaining time Δt_r is obtained by solving the above equation for the duration variable.

$$\Delta t_r = \frac{C}{\sum_i U_i \cdot A_i} \cdot \ln\left(\frac{T - T_{out}}{T_{next} - T_{out}}\right)$$

The Occupancy model is stochastic. Each of an arbitrary number of occupants repeatedly moves to a uniformly sampled (x, y) location for an exponentially sampled duration of time. An occupant that ends up within a fixed distance from the window may open or close it in an effort to bring the indoor temperature closer to 22°C.

Simulation results produced by the coupled model are shown in Figure 7. The window is opened first to accelerate the increase in indoor temperature when it is around 18°C, and later help reduce it from about 30°C. Observe how the time steps vary to allow indoor temperature changes of exactly 0.5°C. The Weather model in this example reads historical data (the TMY3 data set for Chicago O'Hare, 1987-09-27, was used), but its output is also quantized.

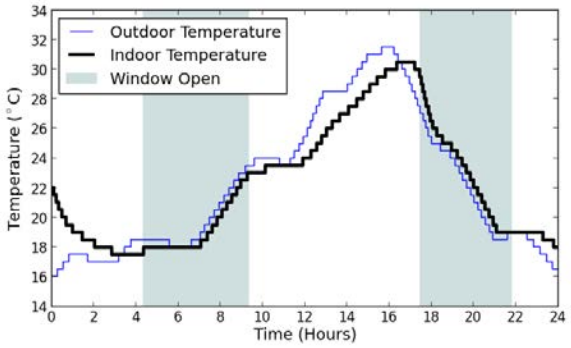

Figure 7: Results for the Weather/Thermal/Occupancy simulation.

With a functional decomposition, most of the complexity ends up in the atomic models. Consider the Thermal model, which must account for one zone and several surfaces. An enhanced version would be considerably more complex, providing separate outputs for each of a building's zones and many of its solid elements. Similarly, the Occupancy model in the example accounts for the window and every occupant. An enhanced version might track all windows, doors, and appliances, and provide outputs for each object manipulation and occupant location change.
With a topological decomposition, the atomic models would be simpler because each would focus on a single type of thermal mass, a single type of manipulable object, or an individual occupant. But whereas the use of functional decompositions is already a practical approach with DesignDEVS or other DEVS-based tools, further research would be needed to address the complexity of topology-based coupled DEVS models for BPS software.

CONCLUSION

The general modeling conventions known as DEVS offer various benefits to researchers striving for a more collaborative approach to BPS software development. From a technological perspective, the use of a DEVS-based simulator alleviates many difficulties otherwise encountered when integrating simulation code. From a mathematical perspective, DEVS does not restrict researchers to a single equation-solving strategy, but allows them to explore several. Finally, from a modeling perspective, domain-specific algorithms implemented by different research teams can be combined via the coupling of DEVS models according to functional decompositions of buildings and their surroundings.

REFERENCES

Beausoleil-Morrison, I., Macdonald, F., Kummert, M., McDowell, T., Jost, R., and Ferguson, A. 2011. The Design of an ESP-r and TRNSYS Co-Simulator. In *Proceedings of the International IBPSA Conference*, Sydney, Australia.

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

Clarke, J. A. 2001. *Energy Simulation in Building Design*. Butterworth-Heinemann, second edition.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. 2003. Taming Heterogeneity - the Ptolemy Approach. *Proceedings of the IEEE*, 91(1):127–144.

Gunay, H. B., O'Brien, W., Goldstein, R., Breslav, S., and Khan, A. 2013. Development of Discrete Event System Specification (DEVS) Building Performance Models for Building Energy Design. In *Proceedings of the Symposium on Simulation in Architecture and Urban Design*, San Diego, CA, USA.

Hensen, J. L. M. 2002. Simulation for performance based building and systems design: some issues and future directions. In *Proceedings of the International Conference on Design and Decision Support Systems in Architecture and Urban Planning*, Eindhoven, Netherlands.

Janak, M. 1997. Coupling Building Energy and Lighting Simulation. In *Proceedings of the International IBPSA Conference*, Prague, Czech Republic.

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

Trčka, M., Wetter, M., and Hensen, J. 2007. Comparison of Co-Simulation Approaches for Building and HVAC/R System Simulation. In *Proceedings of the International IBPSA Conference*, Beijing, China.

Wetter, M. 2009. Modelica-based Modeling and Simulation to Support Research and Development in Building Energy and Control Systems. *Journal of Building Performance Simulation*, 2(2).

Wetter, M. 2011. Co-Simulation of Building Energy and Control Systems with the Building Controls Virtual Test Bed. *Journal of Building Performance Simulation*, 3(4).

Zeigler, B. P., Praehofer, H., and Kim, T. G. 2000. *Theory of Modeling and Simulation*. Academic Press, San Diego, CA, USA, 2nd edition.

Zimmermann, G. 2001. A New Approach To Building Simulation Based on Communicating Objects. In *Proceedings of the International IBPSA Conference*, Rio do Janeiro, Brazil.