A MODULAR BUILDING CONTROLS VIRTUAL TEST BED FOR THE INTEGRATION OF HETEROGENEOUS SYSTEMS

Michael Wetter[1] and Philip Haves[2]
[1]Simulation Research Group, Building Technologies Department,
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA.
[2]Commercial Building Systems Group, Building Technologies Department,
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA.

ABSTRACT

This paper describes the Building Controls Virtual Test Bed (BCVTB) that is currently under development at Lawrence Berkeley National Laboratory. An earlier prototype linked EnergyPlus with controls hardware through embedded SPARK models and demonstrated its value in more cost-effective envelope design and improved controls sequences for the San Francisco Federal Building. The BCVTB presented here is a more modular design based on a middleware that we built using Ptolemy II, a modular software environment for design and analysis of heterogeneous systems. Ptolemy II provides a graphical model building environment, synchronizes the exchanged data and visualizes the system evolution during run-time. Our additions to Ptolemy II allow users to couple to Ptolemy II a prototype version of EnergyPlus, MATLAB/Simulink or other simulation programs for data exchange during run-time. In future work we will also implement a BACnet interface that allows coupling BACnet compliant building automation systems to Ptolemy II.

We will present the architecture of the BCVTB and explain how users can add their own simulation programs to the BCVTB. We will then present an example application in which the building envelope and the HVAC system was simulated in EnergyPlus, the supervisory control logic was simulated in MATLAB/Simulink and Ptolemy II was used to exchange data during run-time and to provide real-time visualization as the simulation progresses.

INTRODUCTION

We discuss the current development of a Building Controls Virtual Test Bed (BCVTB). The BCVTB is a modular, extensible, open-source *software platform* that allows designers, engineers and researchers of building energy and control systems to interface different simulation programs with each other and, in the future, with Building Automation Systems (BAS). Typical applications include the performance assessment of integrated building energy and control systems, either in co-simulation mode or synchronized to real-time, the development of new control algorithms, the testing of control hardware and software in an emulated environment, and the formal verification of control algorithms prior to deployment in a building.

Significant work on using simulation to evaluate the performance of local loop and supervisory control has been performed in the International Energy Agency (IEA) Annex 17 (Lebrun 1992). Control strategies were implemented in simulation and in real, commercial, control hardware coupled via analog interfaces to building envelope and system simulations (Haves et al. 1996; Haves et al. 1991; Kelly, Park, and Barnett 1991; Laitila et al. 1991; Vaezi-Nejad et al. 1991; Wang, Haves, and Nusgens 1994; Peitsman et al. 1994; Decious, Park, and Kelly 1997; Wittwer et al. 2001). The techniques developed in the IEA Annex 17 were further developed in the ASHRAE Research Project 825-RP and resulted in a simulation testbed for control algorithms (Haves et al. 1996). The US National Institute for Standards and Technology has been developing a Virtual Cybernetic Building Testbed (VCBT) that uses the Common Object Request Broker Architecture (CORBA) and BACnet to link computer models for building envelope, HVAC systems, fire and smoke propagation and control hardware. Earlier work conducted by Lawrence Berkeley National Laboratory led to a prototype link between building control systems and EnergyPlus, using SPARK models embedded in EnergyPlus to communicate with the control system via digital/analog converters.

Different building energy analysis programs have also been coupled by several others (Hensen 1999; Lam et al. 2002; Trcka, Hensen, and Wijsman 2006; Zhai and Chen 2005; Trcka, Wetter, and Hensen 2007). In most cases, the coupling is done by a direct link between two simulators. Our approach differs from a direct coupling as we use a modular middleware that allows users to couple to the middleware any simulation program, either locally or remotely over the internet, and BACnet compliant BAS. The middleware allows users to graphically couple simulators and control interfaces, and it provides a library so users can add their own system models directly within the

middleware. Such system models can be used to model physical systems (such as HVAC systems) or communication networks. It also provides models for data processing, such as output analysis, online visualization and reporting. The middleware can simulate systems as fast as possible, synchronized to real-time or at any speed in between. One of our main goals is to make our BCVTB available to other users who can use this platform to link their own simulation programs or control interfaces with little effort. Thus, our work does not attempt to solve the problems of data and process interoperability (Augenbroe et al. 2004; Bazjanac 2004), but we provide a modular software framework that can be used to interface different simulation tools and BAS for run-time data exchange.

INTENDED APPLICATIONS AND REQUIREMENTS

The BCVTB should allow users of EnergyPlus to extend EnergyPlus' capabilities for control simulation and for system simulation. For a researcher who is interested in assessing the performance of new HVAC system architectures that are not implemented in EnergyPlus, a typical use case would be to couple an EnergyPlus model of the building envelop with a modular simulation environment such as Dymola (Brück et al. 2002) that enables the researcher to quickly build a simulation model of a new HVAC system and to couple the two programs for data exchange during run-time. A control engineer who needs to develop a new supervisory control sequence, such as for an active facade of a naturally ventilated building, may develop a control sequence in MATLAB/Simulink and couple MATLAB/Simulink with EnergyPlus through the BCVTB. To verify that a supervisory control algorithm works as intended, a control technician may couple a BACnet compliant BAS to EnergyPlus, which is then used as a virtual representation of the actual building to which the control algorithm will be deployed. This may help reduce commissioning time and eliminate errors in a supervisory control sequence. In future applications, we also envision the use of the BCVTB for local control, but this is not subject of this paper.

Based on the above use cases and on interviews with stakeholders from industry, we deduced the following requirements:

1. The BCVTB should allow users to couple different new clients, i.e., a new simulation program, with code modifications required only in the new client.

2. The BCVTB should be fast enough to be applicable for co-simulation.

3. The BCVTB should be made modular and simulation tool independent so that different clients can be coupled to it. Examples of clients are EnergyPlus, MATLAB/Simulink, simulation environments for Modelica, a BACnet compatible BAS and visualization tools for the online plot of variables.

4. For BACnet operation, the coupling should be fault tolerant in the sense that clients can proceed with their operation even if no updated values are available from BACnet. This situation can occur if communication problems prevent BACnet from sending updated values.

5. The middleware should allow communication with clients over the internet.

6. The middleware should run on Windows XP, Linux and Mac OS X.

IMPLEMENTATION

We implemented the BCVTB using a modular architecture in which a middleware links the different simulation programs and the BAS. To simplify the discussion, whenever we say client we mean a simulation program or a BAS. Using a middleware, as opposed to linking clients directly to each other, allows the coupling of an arbitrary number of clients. It also provides a central point for starting the simulation of all clients, establishing the communication channels, synchronizing the simulation time and stopping the clients. Fig. 1 shows the architecture of the BCVTB where the different clients may be a building energy simulation program, a daylighting simulation program, a building energy control system, a lighting control system and a visualization tool.

Time Synchronization

In the current implementation, we exchange data between the different clients using a fixed synchronization time step. There is no iteration between the clients. In the co-simulation literature, this coupling scheme is referred to as *quasi-dynamic coupling*, *loose coupling* or *ping-pong coupling* (Hensen 1999; Zhai and Chen 2005). For co-simulation, there may be computationally more efficient data exchange scheme (with the communication interval being adaptive depending on the rate of change of variables, and possibly with iteration between the clients if implicit integration schemes are used) but we restrict our initial implementation to this simple data exchange scheme.

Middleware

To implement the middleware, we used the Ptolemy II software (Brooks et al. 2007). Ptolemy II is a Java based software framework developed by the University of California at Berkeley to study modeling, simulation and design of concurrent heterogeneous real-time systems. In Ptolemy II, models can be built by instantiating *actors*.

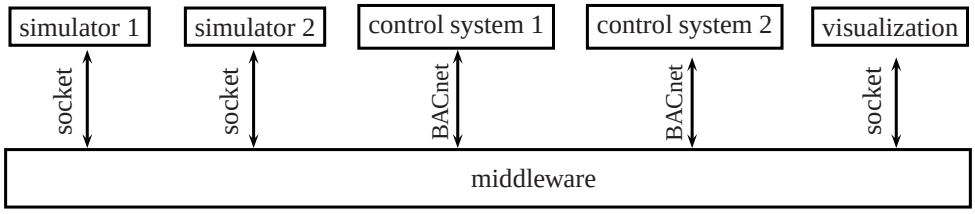


Figure 1: Architecture of the BCVTB with the middleware that connects different clients.

Actors encapsulate the action performed on input tokens to produce output tokens. In the BCVTB, an actor may be a Java class that communicates during run-time with EnergyPlus. Its input token may be a control signal to be sent to an actuator in EnergyPlus, and an output token may be a sensor value received from EnergyPlus. To send output tokens to input tokens of another actor, a user can draw in Ptolemy II's graphical editor a connection line among the actors' ports. The interaction among the actors is defined by a *Model of Computation* (MoC). The MoC specifies the communication semantics among ports. Examples of MoC that are of particular interest for the BCVTB are *Synchronous Data Flow* (SDF) and *Finite State Machines* (FSM). We used SDF for linking different actors that represent a simulation program or a BAS. In the SDF, each actor is fired when a fixed, pre-specified number of tokens is available on each of its input. In this domain, each actor produces a fixed, pre-specified number of output tokens at each firing. Another domain of interest for the BCVTB is the Finite State Machine domain. In this domain, entities are not actors but rather *states*, and the connections among the entities represent transitions between states. FSM are useful for expressing control logics in which different control laws are used depending on the state of the system, such as sequencing control strategies for air-handling units (Seem, Park, and House 1999). The modular extensible structure of Ptolemy II will allow future use of the BCVTB for other research applications, such as analyzing the effect of lost packages or of communication latency in a communication network on the performance of a building control system, or the integration of optimization algorithms for demand response control.

Additions to Ptolemy II

We added to Ptolemy II a new Java package called `LBNLActors`. This package contains the classes `Simulator`, `net.Server` and `util.XMLWriter`. The class `Simulator` is an actor that starts a simulation program, sends its input tokens to the simulation program, receives new values from the simulation program and outputs these values to its output port. The class `Simulator` uses the class `net.Server` for the interprocess communication with the simulation program. The interprocess communication is implemented using the Berkeley socket interface (BSD sockets). In our implementation, `net.Server` is the server and the simulation program is the client. To connect to the server, the client needs to know on what port number it needs to connect. To pass the port number from the server to the client, the class `Simulator` writes an XML file using `util.XMLWriter`, which is then read by the client at program start.

This implementation allows the class `Simulator` to start through a system call any executable, such as a batch file on Windows, a shell script in Mac OS X or Linux, or directly a compiled program such as MATLAB/Simulink.

Library for Clients

We implemented two libraries with C functions that can be used by developers to implement an interface in client programs that allows the client program to connect to and to communicate with the BCVTB. One library provides functions for parsing XML files. This library can be used by clients to parse the configuration file shown in Fig. 5 below using search commands in the XPath language. The other library provides functions for establishing the BSD socket connection and for exchanging data through the BSD socket. We will show in the next section how these libraries can be used to add new clients.

For developers, the BCVTB contains Makefiles to compile the library, the Ptolemy II additions and illustrative examples. The Makefiles can be used on Windows (using Cygwin), Mac OS X and Linux. The library and Ptolemy II additions are documented using the doxygen automatic code documentation system.

Example to Illustrate how to Connect a Client to the BCVTB

We will now show an example to illustrate how to connect a client program to the BCVTB. We consider the configuration where we have two rooms, each with a heater that is controlled by a proportional controller. We will implement the simulation program for the room in a C program and the controller in Ptolemy II.

Let $k \in \{0, 1, 2, \ldots\}$ denote the number of equidistant time steps and let $l \in \{1, 2\}$ denote the number of the room. For the k-th time step and the room number l, let

$T^l(k)$ denote the room temperature and let $u^l(k)$ denote the control signal for the heater. We use for the room temperature the equation

$$T^l(k+1) = T^l(k) + \frac{\Delta t}{C^l}(UA)^l\,(T_{out} - T^l(k)) + \frac{\Delta t}{C^l} Q_0^l\, u^l(k), \tag{1}$$

where Δt is the time interval, C^l is the room thermal capacity, $(UA)^l$ is the room heat loss coefficient, T_{out} is the outside temperature and Q_0^l is the heater's nominal capacity. In (1), we assumed that the communication time step is small enough to be used as the integration time step. If this is not the case, we could of course use a different integration time step or integration algorithm and synchronize the integration time step with the communication time step. The governing equation for the control signal is

$$u^l(k+1) = \min\big(1, \max\big(0, \gamma^l\,(T_{set}^l - T^l(k))\big)\big), \tag{2}$$

where $\gamma^l > 0$ is the control gain, T_{set}^l is the set point temperature and the $\min(\cdot,\cdot)$ and $\max(\cdot,\cdot)$ functions limit the control signal between 0 and 1.

Fig. 3 shows the source code of the implemented client. There are two function calls that interface the client with the BCVTB: On line 2, the function call `establishclientsocket` establishes the socket connection from the client to the middleware. The return value is a descriptor that references the socket. This descriptor is then used on line 11 as an argument to the function call `exchangewithsocket`. This function writes data to the socket and reads data from the socket. Its arguments are the socket file descriptor, a flag to send a signal to the middleware (a non-zero value means that the client will stop its simulation) and a flag received from the middleware (a value of 1 indicates that no further values can be written to or read from the socket by the client). The remaining arguments are the array lengths and the array data to be written to and read from the middleware. There is a double array, an integer array and a boolean array. Even though the C language lacks a true boolean type, we added a vector of boolean values as clients that are implemented in other languages that may want to exchange boolean variables.

For this example, we implemented the controller directly in the middleware, using actors from the Ptolemy II library; however, the controller could as well be implemented in MATLAB/Simulink or in a user written program that communicates through a BSD socket similarly to the C client above. Fig. 2 shows the system diagram with the actor for the controller and the actor that interfaces the simulation program.

In Fig. 4 we show the sequence of data exchange between the clients and Ptolemy II. In this schematic, we assumed two clients, but more clients are possible if needed. The figure shows the function calls

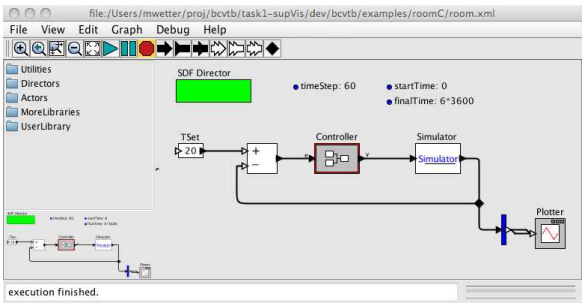


Figure 2: System diagram that couples the simulation program for the rooms and the controller in Ptolemy II.

(rectangles), the data exchange between the clients and Ptolemy II (straight arrows), the data exchange inside Ptolemy II (circular arrows) and the simulation time in the clients and in Ptolemy II. The simulation times are not to scale. The hatched boxes are calls to `exchangewithsocket`. Once a client has initialized its data, it calls `exchangewithsocket` to write its initial values to Ptolemy II, as indicated by the arrow y_0 for client 1. Ptolemy II will send these initial values to the other Ptolemy II actors, which may include client 2. This exchange within Ptolemy II is indicated by the first circular arrow. Then, the time integration starts: Ptolemy II sends data that include initial conditions of other clients to the sockets and the clients receive them. This is illustrated by the arrow labeled u_0 where data is sent to client 1 which still waits until its first call to `exchangewithsocket` returns. Now, client 1 computes $y_1 = f(u_0)$ by advancing the simulation time from $t = 0$ to $t = 1$. Next, client 1 may call its output report routines and then call `exchangewithsockets` to write y_1 to Ptolemy II and to receive u_1 from Ptolemy II. Once u_1 has been received, the computation of $y_2 = f(u_1)$ can start. This exchange scheme continues until Ptolemy II reaches its final time. Then, Ptolemy II writes a flag with a value of 1 to the client to indicate that the last time step has been reached and, hence, that the client won't receive any further data.

EnergyPlus Interface

In a prototype version of EnergyPlus, we implemented an interface to the BCVTB. The interface allows the BCVTB to write to a special version of a `DAYSCHEDULE` object, which we named `DAYSCHEDULE:DYNAMIC`, and to read any `REPORT VARIABLE` that has been set up in the EnergyPlus input data file (idf-file). We also plan to implement a schedule based on the `SCHEDULE:COMPACT`, but this is not yet completed. The idf entry to write to a day schedule with name `SP-TC` and to allow reading the room temperature is as follows:

```
1 DAYSCHEDULE:DYNAMIC,
2  SP-TCooling,    ! Name
```

```
// Establish the client socket
const int sockfd = establishclientsocket("simulation.cfg");
if (sockfd < 0){
  fprintf(stderr,"Error: Failed to obtain socket file descriptor.\n");
  exit((sockfd)+100);  }
// Simulation loop
while(1){
  // assign values to be exchanged
  for(i=0; i < nDblWri; i++)  dblValWri[i]=TRoo[i];
  // Exchange values
  const int retVal = exchangewithsocket(&sockfd, &flaWri, &flaRea,
      &nDblWri, &nIntWri, &nBooWri, &nDblRea, &nIntRea, &nBooRea,
      &simTimWri, dblValWri, intValWri, booValWri,
      &simTimRea, dblValRea, intValRea, booValRea);
  // Check flags
  if (retVal < 0){
    printf("Simulator received value %d when reading from socket. Exit simulation.\n", retVal);
    closeipc(&sockfd);   exit((retVal)+100);  }
  if (flaRea == 1){
    printf("Simulator received end of simulation signal from server. Exit simulation.\n");
    closeipc(&sockfd);   exit(0);  }
  if (flaRea != 0){
    printf("Simulator received flag = %d from server. Exit simulation.\n", flaRea);
    closeipc(&sockfd);  exit(1);  }
  // Assign exchanged variables
  for(i=0; i < nRoo; i++)
    y[i] = dblValRea[i];
  // Compute new state and update time x(k+1) = f(y(k))
  for(i=0; i < nRoo; i++)
    TRoo[i] = TRoo[i] + delTim/C[i] * ( UA * (TOut-TRoo[i] ) + Q0Hea * y[i] );
  simTimWri += delTim; // advance simulation time
} // end of simulation loop
```

Figure 3: Code snippet that shows the integration of simulator in the BCVTB.

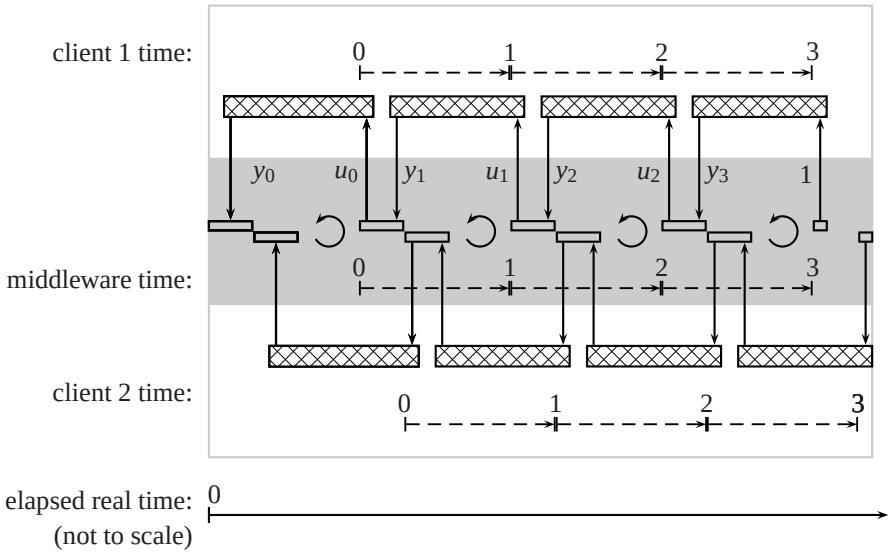


Figure 4: Time synchronization and function calls between the Ptolemy II middleware and two clients.

```
 Temperature,    ! ScheduleType
 24;             ! Initial value,
                 ! used during warm-up

DAYSCHEDULE:DYNAMIC,
 SP-THeating,    ! Name
 Temperature,    ! ScheduleType
 20;             ! Initial value,
                 ! used during warm-up

Report Variable,
 ZONE SOUTH,                ! Key_Value
 ZONE/SYS AIR TEMPERATURE, ! Variable_Name
 timestep,             ! Report_Frequency
 ReportSch;            ! Schedule_Name
```

The schedule type in `DAYSCHEDULE:DYNAMIC` is treated in EnergyPlus the same way as a `DAYSCHEDULE` object. Next, a user needs to provide an XML file that allows the BCVTB interface to connect the variables in the array of double values to the corresponding EnergyPlus variables. Fig. 5 shows an example of such an XML file. In this file, the value of the attribute `source` specifies what software computes the variable.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BCVTB-variables>
  <variable source="EnergyPlus">
   <EnergyPlus name="ZONE SOUTH"
       type="ZONE/SYS AIR TEMPERATURE"/>
  </variable>
  <variable source="Ptolemy">
    <EnergyPlus dayschedule="SP-TCooling"/>
  </variable>
  <variable source="Ptolemy">
    <EnergyPlus dayschedule="SP-THeating"/>
  </variable>
</BCVTB-variables>
```

Figure 5: XML file for client configuration.

Simulink Interface

We also implemented a Simulink block that enables a Simulink model to exchange data with our `Simulator` class in Ptolemy II. Fig. 6 shows how to link a Simulink controller model (block labeled "controller" in the figure) with a model that handles the interface with Ptolemy II (block labeled `socketIO`). Input to the block `socketIO` are a vector containing the control signals, such as setpoints and actuator values, and a trigger signal. If the trigger signal is 0, then the block will not be called during the simulation. This enables a model builder to test and debug the control algorithm in isolation from the plant model. Output of the block `socketIO` is a vector of sensor values that has been received from Ptolemy II. Since

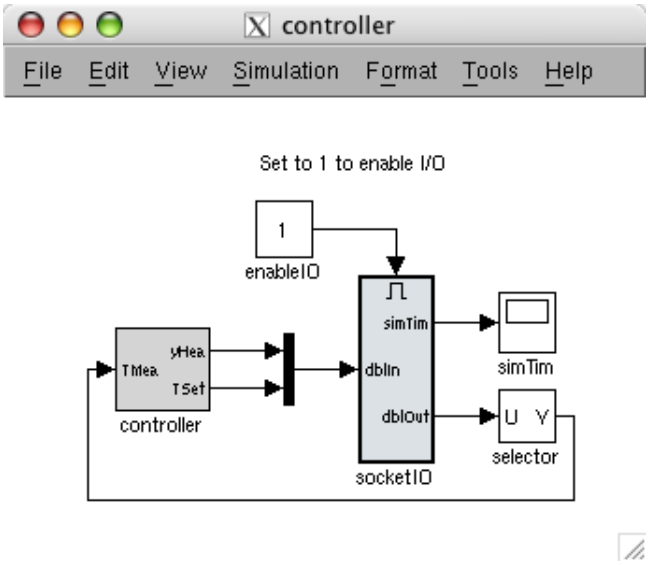


Figure 6: Simulink model that interfaces a Simulink controller model (left grey box) with a model that interfaces Ptolemy II (right grey box).

MATLAB's code generation tool, which we used to interface our C socket library requires MATLAB function outputs to have a fixed array size, the output vector of `socketIO` has typically more elements than required. The block `selector` selects the array elements needed by the controller. If more elements are needed than are currently allocated in the `socketIO` interface, a user can increase the value in a MATLAB script and recompile the MATLAB/Simulink socket library using a Makefile that is part of the BCVTB environment.

EXAMPLE COSIMULATION OF ENERGYPLUS AND SIMULINK

We now present an example in which we linked EnergyPlus and MATLAB/Simulink to the BCVTB. EnergyPlus simulates the building including its natural ventilation and MATLAB/Simulink simulates a controller that determines the window opening positions. Both programs exchange via Ptolemy II data every 1 minute of simulation time. In EnergyPlus, we use two report variables for the room air temperatures of two rooms through which cross ventilation occurs, and we use a report variable for the outside drybulb temperature. These three temperatures are input into the Simulink controller, together with the current clock time. The controller determines the room air setpoint temperature, using different values for day and night and a 2 Kelvin deadband between opening and closing the windows. The control logic is as follows: For some time step $k \in \mathbb{N}$, let $T_s(k)$ denote the setpoint temperature, let $T_r(k)$ denote the bigger of the two room tem-

peratures and let $T_o(k)$ denote the outside drybulb temperature. If $T_r(k) > \max(T_s(k), T_o(k))$, then windows are allowed to be open. Otherwise they are closed. There are three opening positions for the windows: For a constant gain $\gamma = 0.5$, the controller computes the control signal $y(k) = \gamma(T_r(k) - T_s(k))$. For $0 \leq y(k) \leq 1$, one window group is open, for $1 < y(k) \leq 2$ two window groups are open and for $2 < y(k)$ all windows are open. This control logic is implemented in a Simulink block similar to the block labelled `controller` in Fig. 6.

Fig. 7 shows the temperature trajectories and the window positions. In the first few days, there is significant chattering, with the window opening and closing in a 15 minute long limit cycle (lower graph). This would be unacceptable for building occupants and would wear out the actuators. Hence, a more sophisticated control algorithm should be used as was implemented in the actual building. A better control algorithm may involve modulating the window opening to avoid the bang-bang control that led to the limit cycle. In the future we anticipate to use the EnergyPlus Energy Management System module, currently under development at the National Renewable Energy Laboratory (Ellis, Torcellini, and Crawley 2007), to write continuous actuator signals from the BCVTB control signals to actuators in EnergyPlus, thereby modulating the window opening position.

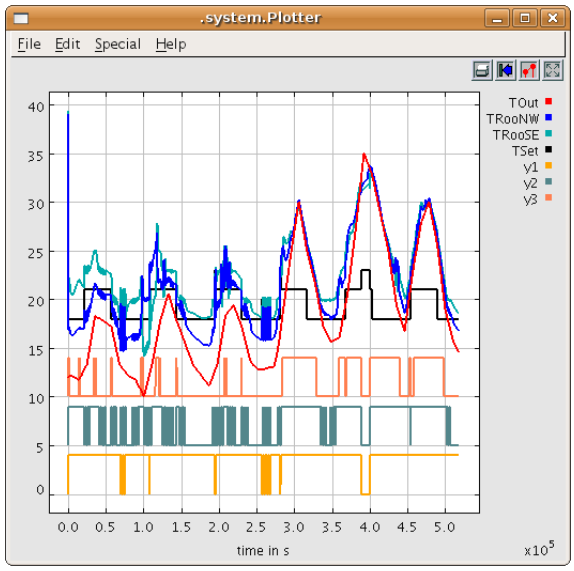
CONCLUSIONS

Using the Ptolemy II modeling and simulation environment as a middleware for the BCVTB allowed us to create a modular environment to which different simulation programs or building control systems can be coupled. It also offers users the possibility to implement system models directly in Ptolemy II, for example, to model physical systems or control systems using different models of computation, such as continuous time, synchronous data flow or finite state machines. Ptolemy II also contains libraries of component models that can be used for data processing; example applications include visualization or transformation of data that is exchanged between different clients.

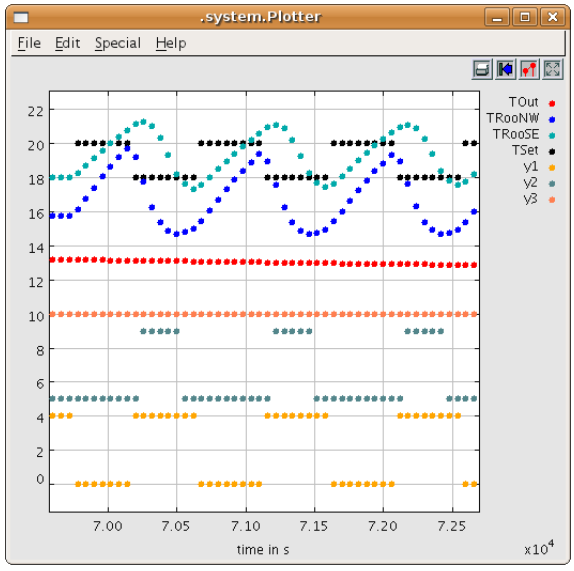
The interface for simulation clients that we added in the form of a Ptolemy II model allows users to add additional simulation programs to the BCVTB by adding two function calls to the new simulation program; one call for establishing a communication connection with Ptolemy II and one call for exchanging data during run-time.

ACKNOWLEDGMENT

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.



(a) 5 days plot during summer



(b) 45 minutes magnification during night

Figure 7: Ptolemy II's online plot showing the outside air temperature (red), the room air temperatures (blue and green), the room set point temperature (black) and the window openings (the upper lines indicate open and the lower lines indicate closed windows) during five summer days (upper graph) and during 45 minutes at night (lower graph).

REFERENCES

Augenbroe, Godfried, Pieter de Wilde, Hyeun Jun Moon, and Ali Malkawi. 2004. "An interoperability workbench for design analysis integration." *Energy and Buildings* 36 (8): 737–748 (August).

Bazjanac, Vladimir. 2004. "Building energy performance simulation as part of interoperable software environments." *Energy and Buildings* 39 (8): 879–883 (August).

Brooks, Christopher, Edward A. Lee, Xiaojun Liu, Steve Neuendorffer, Yang Zhao, and Haiyang Zheng. 2007. "Ptolemy II – Heterogeneous Concurrent Modeling and Design in Java." Technical Report, University of California at Berkeley.

Brück, Dag, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. 2002, March. "Dymola for Multi-Engineering Modeling and Simulation." Edited by Martin Otter, *Proceedings of the 2nd Modelica conference*. Modelica Association and Deutsches Zentrum fur Luft- und Raumfahrt, Oberpfaffenhofen, Germany, 55–1 – 55–8.

Decious, Gaylon M., Cheol Park, and George E. Kelly. 1997. "A Low Cost Building/HVAC Emulator." *HPAC Heating/Piping/AirConditioning*, January, 188–193.

Ellis, Peter G., Paul A. Torcellini, and Drury B. Crawley. 2007. "Simulation of Energy Management Systems in EnergyPlus." Edited by Jiang Yi, Zhu Yingxin, Yang Xudong, and Li Xianting, *Proc. of the 10-th IBPSA Conference*. International Building Performance Simulation Association and Tsinghua University.

Haves, P., A. L. Dexter, D. R. Jorgensen, K. V. King, and G. Geng. 1991. "Use of a Building Emulator to Develop Techniques for Improved Commissioning and Control of HVAC Systems." *ASHRAE Transactions* 97, no. 1.

Haves, P., L. K. Norford, M. DeSimone, and L. Mei. 1996. "A Standard Simulation Testbed for the Evaluation of Control Algorithms & Strategies." Final report 825-RP, ASHRAE, Atlanta, GA.

Hensen, Jan L. M. 1999. "A comparison of coupled and de-coupled solutions for temperature and air flow in a building." *ASHRAE Transactions* 105 (2): 962–969.

Kelly, G. E., C. Park, and J. P. Barnett. 1991. "Using emulators/testers for commissioning EMCS software, operator training, algorithm development, and tuning local control loops." *ASHRAE Transactions* 97, no. 1.

Laitila, P. K., R. O. Kohonen, K. I. Katajisto, and G. K. Piira. 1991. "An emulator for testing HVAC systems and their control and energy management systems." *ASHRAE Transactions* 97, no. 1.

Lam, K. P., A. Mahdavi, S. Gupta, N. H. Wong, R. Brahme, and Z. Kang. 2002. "Integrated and distributed computational support for building performance evaluation." *Advances in Engineering Software* 33 (4): 199–206.

Lebrun, Jean. 1992. "Annex 17." Final report, International Energy Agency.

Peitsman, H. C., S. W. Wang, P. Haves, S. Karki, and C. Park. 1994. "Investigation of the reliability of building emulators for testing energy management and control systems." *ASHRAE Transactions* 100, no. 1.

Seem, John E., Cheol Park, and John M. House. 1999. "A New Sequencing Control Strategy for Air-Handling Units." *HVAC&R Research* 5 (1): 35–59 (January).

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

Trcka, Marija, Michael Wetter, and Jan Hensen. 2007. "Comparison of co-simulation approaches for building and HVAC/R Simulation." Edited by Jiang Yi, Zhu Yingxin, Yang Xudong, and Li Xianting, *Proc. of the 10-th IBPSA Conference*. International Building Performance Simulation Association and Tsinghua University.

Vaezi-Nejad, H., E. Hutter, P. Haves, A. L. Dexter, G. Kelly, P. Nusgens, and S. Wang. 1991, August. "Use of Building Emulators to Evaluate the Performance of Building Energy and Management Systems." Edited by J. A. Clarke, J. W. Mitchell, and R. C. Van de Perre, *Proc. of the IBPSA Conference*. Nice, France.

Wang, S. W., P. Haves, and P. Nusgens. 1994. "Design, construction and commissioning of building emulators for EMCS applications." *ASHRAE Transactions* 100, no. 1.

Wittwer, Christof, Werner Hube, Peter Schossig, Andreas Wagner, Christiane Kettner, Max Mertins, and Klaus Rittenhofer. 2001, August. "ColSim - a new simulation environment for complex system analysis and controllers." Edited by R. Lamberts, C. O. R. Negrão, and J. Hensen, *Proc. of the 7-th IBPSA Conference*, Volume I. Rio de Janeiro, Brazil, 237–244.

Zhai, Zhiqiang John, and Qingyan Yan Chen. 2005. "Performance of coupled building energy and CFD simulations." *Energy and Buildings* 37 (4): 333–344 (April).