Building Simulation 2009

*Eleventh International IBPSA Conference*
**Glasgow, Scotland**
**July 27-30, 2009**

# REAL CONTROLLERS IN THE CONTEXT OF FULL-BUILDING, WHOLE-YEAR SIMULATION

Per Sahlin, Axel Bring and Lars Eriksson
EQUA Simulation AB, Stockholm, Sweden

## ABSTRACT

Study of complex control strategies plays an increasing role in building design. Discrepancy between the intentions of the designer, often expressed as non-formalized control laws, and the as-built implementation is a frequent source of malfunction and energy waste. Two approaches to remedy this problem in the context of equation-based, full-building, whole-year simulation are treated: (1) The availability of Modelica based libraries of functional blocks for typical objects such as integrator, gain, PID-controller and time-delay in a general system simulator enables off-line testing and tuning of realistic control systems. (2) Efficient implementation of algorithmic discrete-time controllers enables direct simulation of complex control algorithms described with the same source code in the simulator as in the real controller. The first approach is discussed and illustrated briefly. While not available in traditional building simulation environments, such block libraries have been available in other simulation domains for several years already. The second approach is treated more thoroughly, since special methods are needed for buildings, where controller sampling interval is much smaller than the typical timescales of the equation-based building model itself. Initial implementation and testing of a new simulation approach is presented.

## INTRODUCTION

Form and construction of building envelope is a natural focus of the early stages of the building design process. Consequently, the first priority of mainstream building simulation tools is support for complex geometries. The ecological benefits of a cleverly configured building shape over a simple-minded one are obvious. However, a conventional building with an optimal shape will only be marginally better. Improved geometry and envelope will not be sufficient to reach the environmental regulation goals that are presently well underway, for example in terms of requirements of zero carbon emissions from new buildings within the next decade.

Moving to production of zero carbon buildings does not seem possible by gradual improvement. We have to change the way we define the concept of a building and its design process. Future buildings will by necessity involve sophisticated combinations of new systems for production, storage, distribution, and emission of clean air, water, light, heat and electricity. Borehole storage, sorption cooling, controlled natural ventilation, phase change materials and micro CHP are just a few of a range of non-standard technologies that will have to be employed in concert.

A thorough analysis of the operation of virtually any existing office building will reveal ample examples of malfunction and energy waste. One will easily find poorly designed systems without even a theoretical possibility of efficient operation, but more commonly, the intentions of the designer have been poorly described or misinterpreted in the control implementation phase and only very basic testing has been done in the commissioning process. Anyone who has written even a simple computer program knows that bugs inevitably appear in the testing and the more rigorous the testing, the more bugs will appear. Yet control programs are often complex and tailored to individual buildings in spite of the fact that the full range of operating conditions will never be tested. With a real building in the loop, time and cost will only allow testing of a small number of possible situations. Since the quality of control of today's relatively simple HVAC systems already is questionable, it is easy to foresee the consequences on the systems of tomorrow. Radical innovation is needed.

With a control object as costly, cumbersome, and slow as a building, thorough testing of algorithms with respect to a real object will never be possible. A single seasonal storage borehole strategy, for example, may take a few years to evaluate. The only option is to rely on a simulation model of the building with all relevant systems that is close enough to reality to allow off-line testing of all important control modes.

Mainstream simulation tools of today provide only rough approximations of a limited number of control systems and time resolution is often much too poor for control design. Frequently, simulation algorithms are based on the very assumption of perfect control. For example, indoor temperature may at times be treated as a given constant in the solution process and the needed heat to maintain this temperature is calculated instead. In a real control scenario, a controller will measure indoor temperature and then decide how

much heat to release. In order to study real control behavior in most traditional building simulators, very fundamental solution principles would have to be altered.

Within an equation-based simulation framework, the behavior of both the physical object and the control system can be modeled with desired fidelity. Continuous-time models of all standard functional blocks allow construction of controllers with the same topology and parameters as those employed in the physical controller. This allows realistic development of any control strategy that can be formulated by basic control blocks. An example of this method is presented below.

In an equation-based environment, each simulated object must be described by equations and functions. Extensions to the basic mathematical repertoire allow treatment of hysteresis and individual discontinuities, but arbitrary "internal states" in simulated objects that are unknown to the solution algorithm are likely to yield poor overall efficiency and robustness. The solver must be able to "see" every equation.

While access to a library of continuous-time functional blocks allows development and tuning of most control schemes, there are two major reasons for simulating complete discrete-time control programs in the context of a simulated building:

1. Some complex control schemes cannot conveniently be expressed exclusively by "extended" equations. Truly algorithmic descriptions are sometimes needed. A drastic example is model predictive control, where a simulation model of the building itself is exercised by the control algorithm in order to find an optimal control scheme.

2. Quality control. To find practical programming errors, a testing environment as close as possible to the as-built situation is desirable. In addition, effects that stem from the limited sampling rate of the real implementation can only be investigated in the correct time scale.

One way of studying the behavior of as-built control programs with respect to a simulated building are hardware-in-the-loop simulations, where a physical controller is interacting with a simulated control object (building). This type of study was for example made in IEA Annex 17 (Karki 1993). In addition to obvious practical difficulties in dealing with a physical controller, this approach may also be cumbersome since experiments normally must be carried out in real time.

A more attractive option is to execute the actual control code faster than real time while retaining the realistic algorithmic and sampling behavior of the physical controller. In its most straightforward implementation, this implies simulation of both controller and building at the sampling rate of the controller,

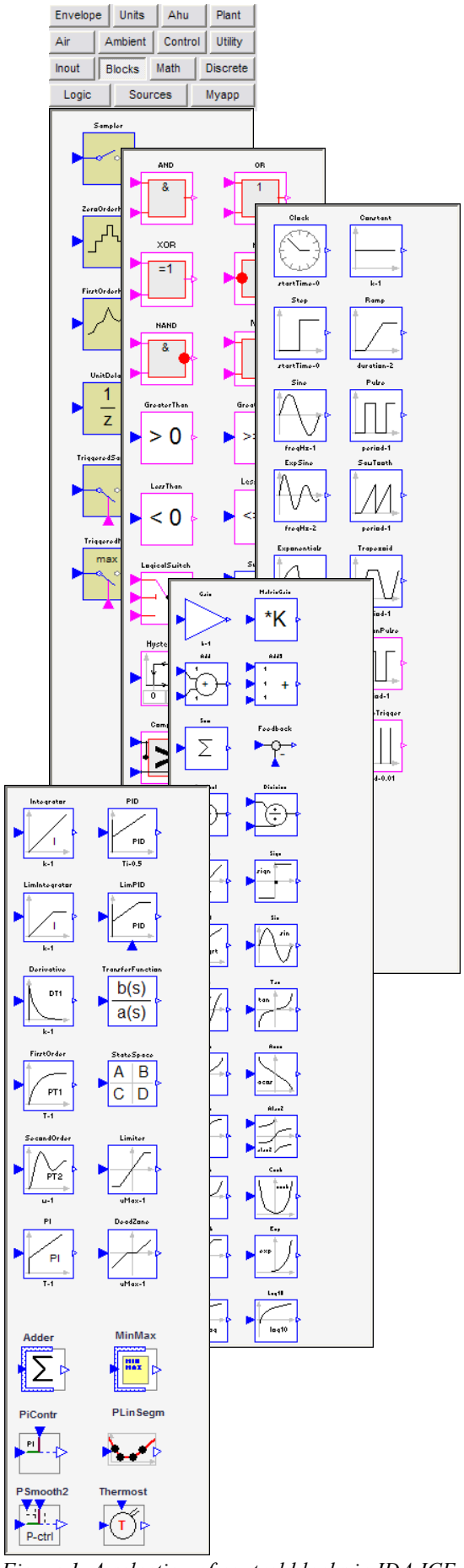


*Figure 1. A selection of control blocks in IDA ICE 4.0.*

i.e. a global simulation time step is taken for each controller sample. For a building, this normally yields excessive simulation times, since a typical controller operates with an unsuitably short step (less than a second). Below, an IDA implementation of discrete-time controllers is presented which enables multi-rate simulation, i.e. a short fixed time step is used for the simulated controller, while a physically motivated much longer (and variable) step is used for the building. Performance comparisons between continuous and discrete implementations of equivalent controllers are presented.

## MODELICA IN BUILDING SIMULATION

The need to model complex and heterogeneous physical systems together with controls is of course not specific to the building domain. Many other industries, such as automotive and chemical process, have similar needs. Numerous attempts have been made to extend the domain of applicability of stand-alone simulators by co-simulation, i.e. to run several independent solvers in parallel while simultaneously synchronizing common variables that occur in more than a single model. In the automotive industry, for example, different groups have traditionally developed stand-alone simulators for the engine combustion, engine motion, drive train, brake system, whole-car dynamic motion etc. It is very natural as a first attempt to try to run these models together rather than to embark into the daunting effort of rewriting them in a more general environment. However, it seems that few truly successful co-simulation projects have been reported and that the focus instead has shifted towards holistic equation-based models, see e.g. (Tiller et al. 2000).

Since the first proposal of the Modelica language in 1997 (Elmqvist et al. 1997), the previously heterogeneous world of equation-based simulation environments has found a focus. After several years of proof-of-concept type projects, the Modelica community ( www.modelica.org ) has now moved on to the development of large base libraries of physical models for a variety of industrial domains. Several international projects such as EuroSysLib and MODELISAR have been started with a significant share of model library development. While Dassault Systèmes' Dymola system is the market leader, sev-

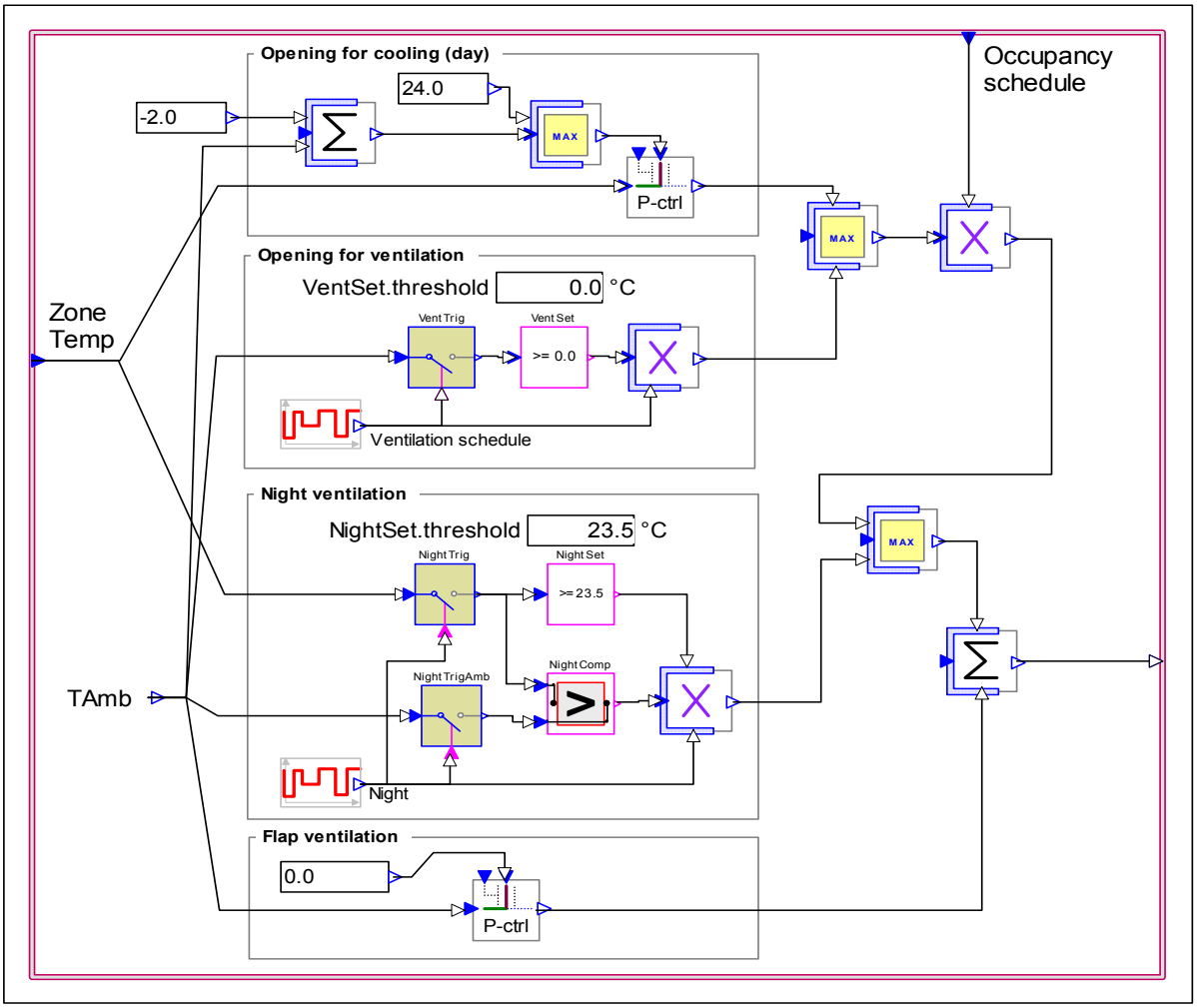


*Figure 2. A controller block diagram in the IDA based VELUX EIC tool*

eral competing commercial Modelica implementations have been released during the last few years and an ambitious open source effort (OpenModelica) is underway.

Modelica has also been applied to building simulation problems. Building envelope model libraries have been described by e.g., Felger et al. (Felger et al., 2002) and Wetter (Wetter, 2006). Since building envelope objects also previously have been successfully modeled in equation-based tools, their description in Modelica is rather straightforward. However, it should be noted that a first version of a model library is useful only to a Modelica expert and that much work remains before such a library can be made useful to the typical building simulation end-user.

Within the IDA Simulation Environment - the base of the recently released whole-building simulator IDA Indoor Climate and Energy 4.0 (IDA ICE 4.0) (EQUA, 2009) - an experimental Modelica translator has been operational since 1999. Although the translator is still not released as a fully supported product, this enables seamless integration of Modelica models into the native NMF based model library. Within the standard delivery of IDA ICE 4.0, the most obvious trace of this is the inclusion of the standard Modelica control block library (Figure 1).

## LIBRARIES OF CONTINUOUS-TIME CONTROL BLOCKS

The control block library in IDA ICE 4.0 has, for example, been used in a model for the automatic window control of the Velux EIC Visualizer (http://eic.velux.com). In Figure 2, the editable schematic of the actual controller is shown.

Four simultaneous window control mechanisms are implemented in the example:

1. "Modulated" window opening to maintain an air temperature setpoint by cooling when the outdoor temperature is lower than the indoor by a given number of degrees. Only active when the building is occupied.
2. If the outdoor temperature is above a given threshold, windows are opened for a given period (typically 10 min), morning, midday and evening. Only active when the building is occupied.
3. If at 22:00, the outdoor temperature is lower than the indoor, and the indoor temperature is above a given threshold, roof windows are left open between 22:00 and 07:00. Only active when the building is occupied.
4. Ventilation flaps of roof windows are permanently open (for background ventilation) when the outdoor temperature is above a certain setpoint.

In the given example, the controller attempts to mimic a usage pattern of a real occupant. Although, this may not be the most common situation, the example illustrates the type of complexity that is commonly implemented in real-life controllers. It is obvious that the full-year consequences of such an algorithm are exceedingly difficult to predict without an appropriate whole-building simulation model. Also clear is that no matter how many hard-coded control algorithms that are included in a traditional simulation program, there will always be a good argument for studying some non-included variant. The only viable solution seems to be to provide the end-user with the means of formulating arbitrary controllers.

Even if the control block library enables end-users to experiment with real complexity controllers, this is of little use if the implementation of the resulting scheme in the actual building by mistake turns out to be something different. A schematic like that of Figure 2 can easily be misinterpreted and implementation of the "typical" control functions can vary slightly in different environments. Hence, the need to formally generate control code based on the simulation model remains.

Automatic generation of controller programs from a simulator representation, requires that standardized languages are used on common control platforms. In industrial automation applications, the IEC 61131-3 language is a common standard. The SimForge (http://trac.elet.polimi.it/simforge/) open source graphical editor allows automatic generation of IEC 61131-3 from a Modelica-based simulation model. Implementation of corresponding functionality in an equation-based building simulator is straightforward.

## DISCRETE-TIME ALGORITHMIC CONTROLLERS

Normally, when discrete-time controllers are simulated within a basically continuous system simulator, the simulation is restarted for each sample of the controller, i.e. the (fixed) timestep of the controller is used for all models. If the dynamics of the physical system are such that this timestep is fairly well suited to resolve relevant transients, this will yield acceptable simulation performance. However, this is rarely the situation for buildings. On average, a suitable timestep for a whole-year full building simulation will be at least a few minutes. In a variable timestep environment, steps of several hours may be taken during the night, when little is happening in the system. Meanwhile, a typical PLC will repeatedly execute its program at full speed rendering a sampling interval on the order of seconds. Buildings and similar objects that are slow with respect to typical sampling times will clearly require special methods. Such a method has been developed for IDA and it will be presented in the next few sections.

### Solver characteristics

The experiments reported here have been made in IDA ICE 4.0. This tool is implemented in the IDA

Simulation Environment, which is a general purpose tool for differential-algebraic simulation.

The numeric solver used in IDA (Eriksson et al.,1992) is a variable step and variable order prediction-corrector solver based on the MOLCOL methods (Dahlquist, 1983), a generalization of the implicit BDF methods. In each normal timestep the future development of all continuous variables is predicted using a polynomial extrapolation. A solution of the system of equations is then calculated by a modified Newton-Raphson method, outlined next.

Analytic component Jacobian matrices are automatically calculated by differentiating the equations with respect to all variables. Analytic Jacobians can be used for most component models; in some cases, however, numeric Jacobians have to be calculated by perturbing the variables of the component.

The global system Jacobian is assembled and factorized. The predicted values are inserted in the equations and residuals are calculated and used as right-hand side when solving the linear system of equations to get a correction vector. The scheme is iterated until residuals and corrections become sufficiently small. Jacobians are not necessarily computed in each timestep, just when convergence is poor.

The predictor-corrector step may fail, either due to lack of convergence, or, because the local truncation error of the difference approximation is deemed too large. This error depends on timestep size and on the order used, and can be expressed in terms of the difference between the predicted and calculated solutions. In either case, the timestep is reduced and a new prediction is calculated. After a successful step, the local truncation error is used to control the order and the tentative length of the next timestep.

When the integration is started, an initial value calculation (IVC) is made to find a start solution to the equations. An IVC will also be made when a discontinuity is met in driving data or an abrupt event is signaled by a component model.

**Interface to individual sampling components**

IDA solver handles component models written in either the Neutral Model Format (NMF) or Modelica. The models are automatically translated to Fortran or C and linked into Windows dynamically linked libraries (DLLs).

To incorporate an arbitrary discrete-time programmable controller in an IDA simulation, it has to meet some basic requirements:

1. It must be written in a language that can be translated to a Windows DLL.
2. It may have internal memory and internal states that should be preserved between activations. However, since the solver will need to rerun sequences of sampling steps, the internal states must be stored in an array that is accessible from the solver.

The interface between solver and sampling component has been implemented as a shell, written in NMF or Modelica. The shell delivers inputs to, and fetches output from the component. It also provides memory space for the internal states of the sampling component.

**Connecting continuous and sampling components**

The solver normally divides the simulated system into three distinct sections: The central aggregate of continuous components (equation-based), an algorithmic section preprocessing input data and providing input to the continuous section, and an algorithmic section performing post-processing of simulation results. In each timestep, the sections are processed in order pre, central, post.

The components making up the system are linked together by links (signal aggregates) that can be directed (causal) or undirected (acausal). Within the continuous section, acausal links are permitted, while in pre and post sections all links must be causal (inputs of one component connected with outputs of another).

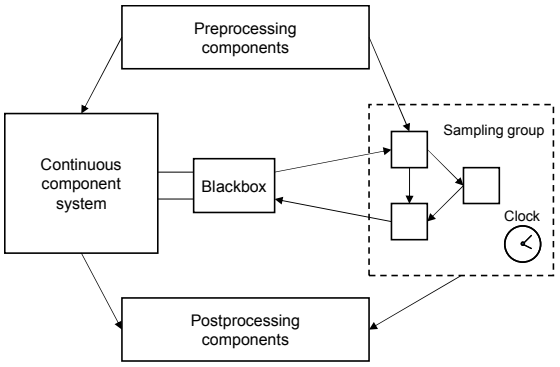


*Figure 3. Component categories.*

When sampling components are added to this setup, they change the pattern. The sampling components will only be connected by causal links. They are allowed to take input from pre and central sections and to deliver output to central and post sections. The pre and post sections will retain their positions first and last in the processing chain, but the central section will be interacting intimately with the sampling components. In this interaction, the sampling components will fetch input from the central section, and in various ways deliver output back to the central section. See Figure 3.

**Groups of sampling components**

In the current implementation, each sampling component is connected to a clock, defining a constant sampling rate. Several components may be combined into a sampling group, provided that they use the same clock, and that they are connected by causal

links into a directed network with a defined execution order. The solver will always activate the group as a whole.

**Typical integration algorithm**

The rationale behind the implementation is the possibility to use multi-step integration, with the central system taking steps much longer than the sampling steps. This is possible if the sampling outputs during longer periods appear as continuous, differentiable signals.

A typical global integration step will progress as follows (see Figure 4). A prediction is calculated for the continuous system, including the variables that are connected to sampling components. In a simple case with a single controller, these variables could be e.g. a temperature sent to the sampling component and a control signal coming back. A more complex case could include several sampling groups, with possibly different sampling rates, and each having multiple inputs and outputs.

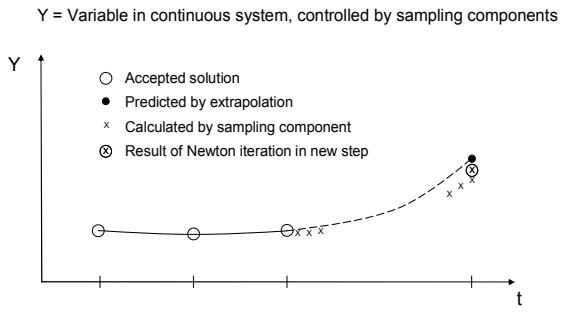


*Figure 4. Solution sequence.*

Integration time is advanced through the global step and the sampling groups are executed at their respective intervals. For each such sampling execution, input signals are interpolated in the prediction. The outputs from the sampling groups are compared with the interpolated predictions for the receiving continuous variables. The sample stepping continues to the end of the global step, unless a too large discrepancy develops underway. In the latter case, the latest sampling execution is cancelled, and the global step is truncated prior to the divergence.

Next, the global system is solved with a Newton iteration, and the accuracy is checked. This scheme is based on the assumption that, as long as the prediction is good enough to provide acceptable truncation error in the continuous equations, the sampling steps, run against the same prediction, can also be accepted as they are, without update to match the corrected continuous solution. Some problems related to this assumption appear, and the remedies taken are discussed below.

**IDA Implementation**

The chosen solution introduces an extra continuous component for each sampling group. This 'black box' component is described as an NMF component and emulates the sampling group, seen from the continuous system. It partakes in all activities pertinent for continuous components; it delivers residuals and Jacobians, the latter obtained by numeric differentiation. This makes a reliable and effective Newton iteration possible.

The component defines one equation for each output from the sampling group, equating this output to the corresponding controlled variable in the continuous system.

All sampling outputs from the group, required for calculation of residuals in these equations, are obtained concurrently by a call of a solver routine. The id of the sampling group is provided as a parameter together with the inputs to the sampling group. The solver uses these data to execute the group, rerunning the latest sampling step with sampling state memory fetched from backup.

The black box component is only active in the solving of the continuous system. The activation of the group when stepping through the global step is done by the solver without reference to the black box.

**Performance**

The performance of the modified implementation is illustrated by some test results presented below. The simulated system is a single office zone with local heating and cooling. Tests were run for a three month summer period with observed climate data.

The cooling room unit was controlled by a PI-controller, implemented both as a continuous model and as a sampling discrete-time algorithm with a rate of 1000 activations per hour. The NMF equations for the continuous version were:

```
E := IF Mode < 0.5 THEN
        (SetPoint - Measure)
     ELSE
        (Measure - SetPoint)
     END_IF ;

OutSignalTemp := k * (E + Integ) ;

OutSignal = IF OutSignalTemp > hilimit THEN
              hilimit
            ELSE_IF OutSignalTemp < lolimit THEN
              lolimit
            ELSE
              OutSignalTemp
            END_IF ;

Integ' = E/ti + (OutSignal - OutSignalTemp)/tt ;
```

where

| | |
|---|---|
| SetPoint | Reference signal |
| Measure | Input signal |
| E | Control error |
| Integ | Integrator term |

| | |
|---|---|
| OutSignal | Control signal |
| OutSignalTemp | Control signal (temp) |
| k | Gain parameter |
| ti | Integration time in seconds |
| tt | Tracking time in seconds |
| mode | Control mode:<br>0= heating type control,<br>1= cooling type control |
| hilimit | High limit for OutSignal |
| lolimit | Low limit for OutSignal |

In the discrete version, the differential equation for the Integ term was instead solved locally:

```
IntegPrim := E/ti + (OutSignal - OutSignalTemp)/tt ;
Integ := Integ + h*IntegPrim;
```

where

| | |
|---|---|
| IntegPrim | Integrator derivative |
| h | Sampling interval |

In the discrete version, the Integ variable is declared as an NMF assigned state, which means that it is memorized between evaluations.

The cases with the sampling controllers were run once with the global timestep equal to that of the controller, and once with multi-step integration. The outcome is summarized in Table 1 for the three cases:

A) Continuous controller (function block)

B) Sampling controller with multi-rate integration (new method)

C) Ditto, solve global system each sampling step (conventional discrete method)

The results show that the sampled controller implementation is less efficient than a continuous ditto, but the time reduction of the new method exceeds 98%.

Figures 5 and 6 show the zone air temperatures for

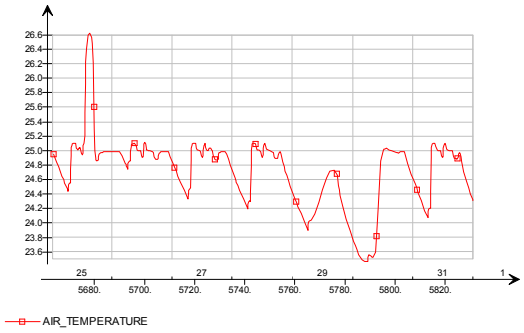


*Figure 5. Air temperature for Case A*

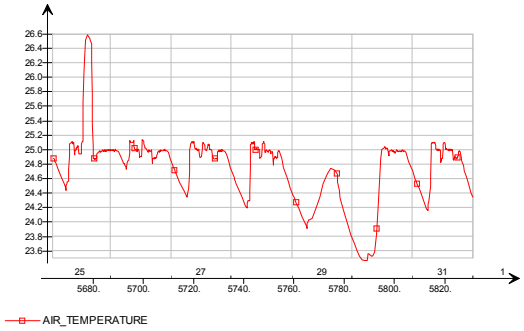


*Figure 6. Air temperature for Case B*

Cases A and B during the last week of simulation. The cooling setpoint, 25C°, is not always met due to limited cooling power. A slight ripple, reflecting the selected tolerance, can be seen in the discrete case.

Similar tests have also been done with on-off controllers, to investigate the performance for discrete state control signals. These results are equally satisfactory, actually showing even smaller penalties for the discrete-time implementation (case B vs. A).

**Short step regime**

The discussion so far focuses on the continuous behavior of the sampling systems. An entirely different scenario appears, if the global timestep happens to be shorter than the sampling steps. This may very well happen, when an abrupt change in the central system or in its inputs triggers fast transients.

Now, the activity of a sampling group can no longer primarily be regarded as smoothly incorporated in a continuous long-term progression. Rather, its discrete

*Table 1. Performance tests, PI-controller*

| | A | B | C |
|---|---|---|---|
| Number of variables | 2 187 | 2 193 | 2 193 |
| Number of steps | | | |
| global successful | 6 946 | 10 540 | 2 209 400 |
| global total tried | 28 035 | 42 546 | 2 216 402 |
| sampling successful | 0 | 2 208 000 | 2 208 000 |
| sampling total tried | 0 | 2 688 241 | 2 209 543 |
| Integration time [s] | 11 | 21 | 1 704 |

nature comes to the fore. When some short global steps have been taken, and the time to activate a sampling group is reached, whatever output it produces will have to be accepted. If they represent a discontinuous change, the solver will make an IVC and then resume global integration. If they appear small, the global integration will have opportunity to increase step length and return to the normal long step regime.

## CONCLUSIONS AND FURTHER WORK

The impact of control design on building performance is often underestimated. In most simulation studies, highest priority is given to building envelope, followed by system operation, while control performance is normally treated with gross simplifications. It is difficult to find other reasons for this situation than the present capabilities of mainstream simulation tools.

In an equation-based simulator, access to libraries of continuous-time control blocks enables accurate simulation of a large range of realistic controllers. Modeling effort can be guided by physical motivation rather than tool capability. Code and settings for actual physical controllers may be automatically generated from the block diagrams and the risk of errors in the implementation process can thereby be reduced.

Unfortunately, block diagram based control descriptions are not always practical. Complex control algorithms are often more succinctly described using an algorithmic language, i.e. by "free programming." Examples of constructions that are cumbersome to realize in a block diagram setting are iterations, extensive rule based evaluation and time-averaging.

A method has been developed that allows efficient simulation of discrete-time algorithms in an equations-based context. The method has been tested on small but realistic examples with satisfactory results.

The next step will be to confront the new method with more complex controllers to ensure its general applicability. At some not too distant future, it will also be natural to develop a simulator-based control design environment, where actual controller code can be developed, tested and deployed to physical devices. However, the attractiveness of such a design tool depends on the wide proliferation of non-proprietary, standardized input description methods (languages) such as IEC 61131-3 or BACNET.

## ACKNOWLEDGEMENT

A significant part of this work has been done within the framework of the EU-funded I3CON (Industrialised, Integrated, Intelligent Construction) project: 26771 (www.i3con.org ).

## REFERENCES

Dahlquist, G. 1983. On One-leg multistep methods, TRITA-NA-8301, Royal Institute of Technology, Stockholm, Sweden

Elmqvist, H., Boudaud, F., Broenink, J., Brück, D., Ernst, T., Fritzson, P., Jeandel, A., Juslin, K., Klose, M., Mattsson, S. E., Otter, M., Sahlin, P., Tummescheit, H., Vangheluwe, H., 1997. Modelica™ - A Unified Object-Oriented Language for Physical Systems Modeling, Version 1, Sept., 1997 (www.modelica.org )

EQUA Simulation AB, 2009, IDA Indoor Climate and Energy 4.0 – User's Guide

Eriksson, L., Söderlind, G., Bring, A., 1992. Numerical Methods for the Simulation of Dynamical Modular Systems, ITM Report, 1992:2, Swedish Institute of Applied Mathematics, Gothenburg, Sweden

Felgner, F., Agustina, S., Cladera Bohigas, R., Merz, R., Litz., L., 2002. Simulation of Thermal Building Behavoir in Modelica. Edited by Martin Otter. Proceedings of the 2nd Modelica conference, 147-154. Modelica Association and Deutsches Zentrum fur Luft- und Raumfahrt, Oberpfaffenhofen, Germany

Karki, S. (ed.)., 1993. Development of Emulation Methods, Research Notes 1514, 1993, 134 pp. + app., VTT, Laboratory of Heating and Ventilation, Finland

Nytsch-Geusen, C., Nouidui, T., Holm, A., Haupt, W., 2005. A hygrothermal building model based on the object-oriented modeling language Modelica. Proceedings of the Ninth International IBPSA Conference, Volume 1. International Building Performance Simulation Association and Ecole Polytechnique deMontreal, 867–876, Montreal, Canada

Tiller, M., P.Bowles, H.Elmqvist, D.Brück, S. E.Mattson, A.Möller, H.Olsson and M.Otter 2000. Detailed Vehicle Powertrain Modeling in Modelica. Modelica Workshop 2000 Proceedings, pp.169-178

Wetter, M., 2006. Multizone building model for thermal building simulation in Modelica. Proceedings of the 4th Modelica conference, Vienna, Austria