Building Simulation

Seventh International IBPSA Conference
Rio de Janeiro, Brazil
August 13-15, 2001

A NEW APPROACH TO BUILDING SIMULATION BASED ON COMMUNICATING OBJECTS

Gerhard Zimmermann
University of Kaiserslautern
67653 Kaiserslautern, Germany

ABSTRACT

Classical building simulators are typically based on global systems of differential equations that model the physical reality and are numerically solved at runtime. In this paper we propose a new approach. Physical components of buildings, such as walls and spaces, are modeled as computational objects that individually solve the appropriate physical equations at runtime and exchange changes of surface values, such as temperatures, when necessary. Because the object structure is derived directly from the building structure and because generic objects are reused, simulators can be generated with very low effort. Based on object oriented modeling and automatic code generation, new physical effects can be easily integrated. As calculations are triggered by changes, very short real-time responses are achieved if necessary. Also, fast and accurate responses to external events can be guaranteed. Both features are necessary for real-time tests of building automation systems.

INTRODUCTION

Today, building simulators are more than just energy consumption and efficiency calculators. The inclusion of light, sound, and thermal comfort extend the possible applications for architects and building engineers [Mahdavi 96]. Another application is prototyping and test of building control systems, which typically are distributed computer programs. Furthermore, simulators can be part of such systems.

We examined existing building simulation approaches in respect to the last two applications. We found simulators based on numerical equation solvers [Dymola] and list- or object-based front ends that translate the building structure and its components into sets of equations [Diehl and Litz, 99]. Some use object-oriented programming [Mahdavi, 96]. None of them is directly applicable to testing or can be used as a part of building control systems because of missing control interfaces and real-time features, insufficient time resolution, or problems with the integration of discrete events, which are caused by inhabitants, failures, and actions of the control system.

Going back to the physics of buildings we modeled building components, such as wall elements, windows, rooms, air volumes, radiators etc., as autonomous objects that have topological and aggregation relations and that exchange value changes, e.g. of surface temperatures, radiation, percentage of openings. Changes are only transmitted when necessary. This closely models what happens in reality. Within the autonomous objects, physical effects are modeled in the classical way by solving the physical equations numerically, e.g. the Fourier equations for the heat flow through a layered wall. No global system of differential equations as in “classical” building simulators has to be solved at runtime. Instead, simple local calculations suffice. The problem we had to solve was the partitioning and modeling of a strongly coupled physical system in such a way that stable and physically correct solutions with minimal computational effort result. By adjusting the minimal deltas for change propagation, accuracy can be traded for speed. Time resolution is dynamically adjusted to the first derivative of changes and to guarantee numerical stability. This results in excellent dynamic responses to changes.

We can show that our approach works well for thermal properties including simple air exchange and solar radiation effects. We have also included simple daylight and artificial lighting effects. The opening of doors and windows and the action of switches and motion detectors by persons are easily integrated into the simulator. Control system interfaces are provided by sockets, based on the TCP/IP protocol. The simulation approach has been tested for a floor with 27 rooms with an occupancy controlled building automation system for heating and lighting.

One major feature of our approach is the ability to generate a simulator program for a given building for the specified physical effects from scratch in a very short time. We have tried two approaches. In the first, the building components and the structure are semi-automatically mapped into Smalltalk classes. The functionality of the classes (implemented by methods) is realized by using *design patterns* from a simulation specific pattern catalogue and by automatic code generation [Schütze et al., 99]. The result is a complete Smalltalk program that simulates the given building. The second approach is based on the abstract modeling of building objects as processes and their functionality as state transition diagrams. Simulated physical values

are passed between components by signals with parameters. The modeling language is SDL [Olsen et al., 94] and the models are automatically translated into executable C-code for a runtime system that provides process scheduling. This approach will be shown in detail. With both approaches real-time execution with the option to accelerate the execution 10 to 100 times real-time was achieved. A sufficient time resolution for continuous effects, ranging from 1s to 1h, for the 27 room floor was observed. Discrete events were handled immediately.

THE SIMULATION MODEL

Buildings are composed of a relatively small number of different *types* of components. In our experience this number is less than 100. In a large building many instantiations of those components with different parameter settings are composed. Depending on the component type, specific physical models for heat and humidity flow and storage, air flow, radiation and sound transmission and reflection apply.

Some components such as luminaires, fans, valves, windows, or shades can be controlled manually or automatically. These are the actuators. Other components, the sensors, measure physical quantities. Both types build the interface to control systems.

All components together form a coupled physical system. In reality, coupling only occurs between adjacent components. Thus the topology of the building determines the physical relations between components.

Coupling is achieved by interface variables. Interfaces are typically related to those surfaces of components that are directed towards adjacent ones. For example, a wall element has surface temperature and heat flow through the surface as thermal variables. Similar variables can be defined for radiation, humidity, or sound. Physical laws determine which variables of adjacent components correspond to each other.

If we enforce that corresponding interface variables of adjacent components have the same value at all times, we arrive at the typical mathematical model of one system of coupled equations that is solved numerically at given time intervals.

If, on the other hand, we allow for temporarily diverging values, we arrive at an iterative approach. The normally simple component models are solved separately, using the last known interface values of all adjacent components as inputs. The iteration continues during a time interval until the interface values converge. Iterative approaches are also used in classical numerical equation system solvers in the case of nonlinear equations.

In a third approach, convergence is not enforced. The procedure is the same as in the iterative approach, but no iterations take place during a time interval. New values are only calculated at every time step, using interface values of the last time step. This occurs in explicit numerical methods for solving systems of parabolic differential equations. By making the time intervals small enough, the errors can be made small and instabilities can be avoided.

The third method can be seen as a delayed interface value exchange. Such delays are not unnatural. If, for example, we model an air volume by a homogeneous average air temperature, a delayed change of the surface temperature on one side as a reaction to a change at the opposite side of the volume would even enhance the models dynamic accuracy.

This example shows that the accurate numerical solution of a model does not guarantee correct results if the model does not reflect important details of the reality. The question of accuracy will be treated in the sectinon after the next.

All three solution methods are used in our object based approach. Especially the latter two methods show how the physical models and the mathematical solutions can be assigned to components or objects in the computational models. The exchange of interface values between objects can be achieved in different ways. We have chosen asynchronous message passing between autonomous objects. In the next chapter we will explain how this is realized.

One important aspect of simulators is the treatment of time. Most input and output variables are time dependent. For energy calculation it is sufficient to increment time during simulation according to the speed of the computation. Increments typically correspond to one hour intervals. This results in the possibility to simulate a whole year in the order of hours, depending on the complexity of the simulated object.

For the test of control systems we need a better control of time. *Real-time* in this context means that the simulation advances time synchronously to wall clock time. In addition, sufficient time resolution is required. For building control a resolution up to 1ms might be required. Only by meeting real-time requirements, the correct reaction of control systems can be tested. Testing speed can be enhanced if the control system can execute with accelerated real-time. This is possible in many cases by running the control system during tests on much faster computers than the target hardware. For this purpose, the building simulator must execute with exactly the same accelerated real-time.Therefore, the upper limit of possible acceleration factors is one of the properties we have to determine.

SIMULATOR OBJECT STRUCTURE

One of the features of objects in our object-based approach is *information hiding*. Most of what occurs within an object is not visible or accessible to its envi-

ronment. This is true for real building components as well. Interfaces clearly define the access to objects. Therefore, we quite naturally assign building components to objects and component types to object types (also called classes).

A second feature, that is not strongly supported in most object oriented environments, is *object concurrency*. In real buildings on the other hand, all components act and react in parallel. In order to map the real world as closely as possible into a computational model, we model all objects as autonomous processes that communicate and synchronize each other by signals with parameters. Every process can execute a physical or administrative function at its own pace as necessary to achieve correct results. A heavy concrete wall can calculate heat transfer at much larger time intervals than a light door. Elements without storage capacitances or in equilibrium can remain passive until external changes are signaled. Only stability requirements of the numerical method can require calculations. Thus, in such a system of autonomous objects, computing can be reduced to a minimum. On the other hand, reactions to sudden changes can be very fast. Here, the question occurs, what will trigger a change signal. This is related to simulation accuracy and we will come back to it in the next chapter.

Another feature is *aggregation*. This concept relates directly to the component structure of buildings. In the same way as buildings are composed of storeys, composed of spaces, composed of offices, composed of workplaces etc., objects can be composed of objects in a hierarchical fashion. This concept supports the management of complexity by *partitioning* the problem into simple components or objects as for example a homogeneous layer in a wall element and, on the other hand, by allowing *abstraction* from unnecessary details by combining components to larger entities as for example to express homogenous air pressure in a large open space. Partitioning and abstraction support regularity by introducing similar types (component or object) at all levels of the hierarchy.

Partitioning and abstraction are both applied in our simulation approach. Model calculations can be carried out by objects at all levels of the aggregation hierarchy.

Partitioning can follow two concepts. The first is the topological structure, defining a compositional hierarchy. The second is the functional decomposition. For example, the component *window* can either be composed of frame, sash, and pane or be composed of heat, thermal radiation, light, air, and sound transmission objects. Both concepts are mixed in our simulators with structural decomposition at higher and functional decomposition at lower levels of the hierarchy.

Aggregation hierarchies are instance hierarchies, building *parts* trees of all objects. These trees can become rather large during modeling. A possible abstraction is the use of acyclic graphs of object types. This representation is much more compact than the tree but instance information is lost. We use a combination of both by annotating the arcs with instance names. We call this a type-instance graph. Figure 1 shows an example.

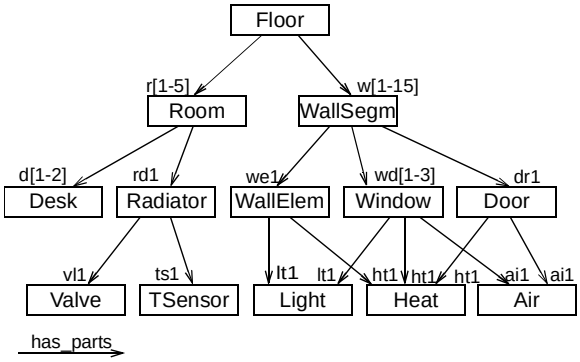


Fig.1: Object type-instance graph example. Types are rectangles, r[1-5] means the instances r1, r2, .. ,r5

In this simplified example, all wall segments are composed of three windows, a door, and a wall element, for example brick masonry. In real examples, different wall segment types would be necessary.

In a simulator every object must be uniquely identifiable for the control and observation of variables. In the type-instance graph the only requirement is that if a type A is composed of several instances of the same type, e.g. type B, the instances of B are named differently. With this condition, every instance can be uniquely named by the concatenation of *instance:type* pairs on the path from the root object to the object in question. Subsets of all instances can be defined by interpreting instance names as regular expressions.

The aggregation hierarchy does not model adjacencies. To express *adjacency relations* we use the regular expressions that identify individual objects or sets of objects. A set can, for example, be used to describe the adjacency of a long room (hallway) to all wall segments of the offices along the hallway with one expression. Signal paths have to be modeled for all these adjacencies to exchange interface values during simulation.

THE ERROR MODEL AND CHANGE PROPAGATION

The purpose of simulation is to model and compute reality as accurately as possible or, more precisely, as accurately as necessary. The latter formulation is important if the goal of simulation is real-time, accelerated real-time or just as fast as possible. The definition of accuracy strongly depends on the application. In many applications, the total energy consumption over a year with a standard weather file has to be with-

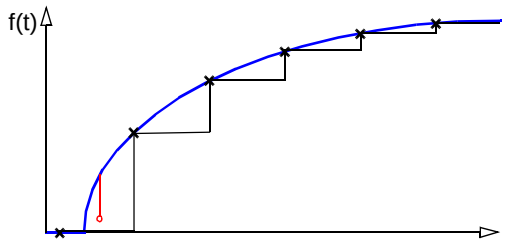


Fig.2: Continuous function and discrete value approximation. The circle shows a small error in *t* resulting in a large error in *f(t)*

in a few percent. In other applications human comfort has to be precisely calculated under all conditions. In our application, the test of building automation systems, the dynamic response of the system as measured by sensors and controlled by actuators and the building environment has to be accurate within given limits.

Therefore, dynamic accuracy or dynamic errors have to be defined. Let us assume a continuous process and a known continuous output function *f(t)* as response to a defined change. The output of the simulation of the process is a series of values *F(i)* at discrete times *t(i)*.

A simple error definition would be the deviation of each calculated value *F(i)* from the correct value *f(t(i))*:

$$e(i) = |F(i) - f(t(i))|$$

The model is not useful because, as Figure 2 shows, very few but accurate samples would result in no error but violate the sampling theorem. On the other hand, small time delays near a steep ascent (circle) can cause very large errors. The latter problem can be reduced by using the Euclidean distance from *F(i)* to the nearest point of *f(t)* as error. But this does not mend the first problem.

Therefore, we defined a more sophisticated error model. It assumes that the time series *F(i)* translates into a step function in the continuos world. The error is defined by the area between the step function and the continuos correct curve:

$$e(i) = \int_{t(i)}^{t(i+1)} |F(i) - f(t)|\,dt$$

The result is that small the time steps result in small errors. It can be seen in Figure 2 that in the case of constant time steps the error increases with the first derivative of the function. This error model makes sense for the assumed application. Fast changes will cause larger reactions in a control system than slow ones and therefore have to be modeled with smaller time steps.

This observation directly results in the need for dynamic time step control. This is a known field of numerics. Unfortunately, most methods require more knowledge about the functions than is available during real-time simulation. Therefore, we control the time intervals such that the first derivatives of the interface values of the computed function together determine the calculation frequency, if this is higher than the frequency of incoming signals.

By appropriate tuning, the error function defined above can be kept within a small range, thus realizing the demand for calculating new values only as often as necessary. The tuning parameters also determine the frequency in general and thus the error limit. This results in the possibility to trade accuracy against speed of simulation.

This demand has another aspect. In order to also reduce the number of signals as far as possible, the sender of a change has to decide if the delta is large enough to be necessary for the receiver to avoid errors larger than the set limit. Although it would be easier for the receiver to decide, this would cause unnecessary signals and also unnecessary wake ups of the receiver object's process. Once a process is running, the model computation can be executed without much more effort. Therefore, every incoming signal results in a new time step at the time of its occurrence.

One of the problems of this type of asynchronous change propagation is that it may result in many input signals at some objects with many adjacent objects that all result from the same event and arrive at nearly the same time. If each of these signals results in an individual computation, many unnecessary computations are performed and unnecessary change signals are sent. The latter can result in a signal avalanche, defeating the goal of minimizing the computational effort. In this asynchronous world an object cannot know how many signals to expect. As a solution in such cases, the object just waits a short time for further signals to arrive and then starts the computation.

As a consequence of this asynchronous behavior, signal bursts and thus computational demands can occur that are larger than what can be handled in real-time or accelerated real-time. In this case change signals may be lost on purpose. As long as this loss results in a small temporary reduction of accuracy it can be tolerated. We call this graceful degradation. We only have to make sure that unique discrete events, such as a person entering a room, are not lost. This mechanism can also be used to automatically trade speed for accuracy without changing the simulator.

THE OBJECT BASED MODELING APPROACH

In our approach the object model with its relations and functions is translated into formal computational models from which simulation code is generated automatically. This is different from object oriented simulator approaches with predefined classes where the object

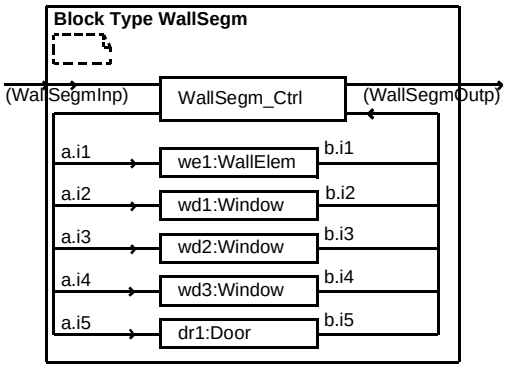


Fig.3: Block type example *WallSegm*

model is only used to generate the instances of the classes. This makes our approach much more flexible and easier to extent to new functionalities.

We use the graph description language SDL [Olsen et al., 94] and the tool set SDT [Telelogic] for editing, code generation, testing, and as a runtime environment. SDL has the advantage of only three well defined, integrated types of diagrams over UML's nine. Full code generation is mature and relatively efficient.

As a first task the type-instance graph of the building to be simulated is transformed into SDL block type diagrams. In order to make this manual step efficient and to support reuse at the modeling level, block type templates are used with very strict interfacing rules. Figure 3 shows the object type *WallSegm* with five instances of three different object types, realizing a part of the type-instance graph in Figure 1. A total of eight similar block type diagrams would model the complete aggregation hierarchy. Leaf nodes are modeled as processes only.

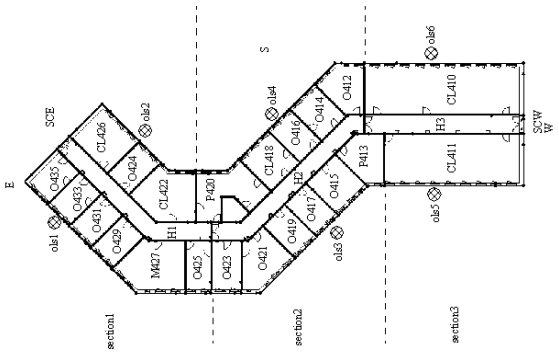
At the top of the diagram the block *WallSegm_ctrl* refers to a process which models the behavior of *WallSegm*. Since the segment is composed of five different areas, for example the brick *WallElem* with its own heat transfer component *Heat*, *WallSegm_ctrl* only computes the sum of all five heat transfers and an average surface temperature. For this purpose *WallSegm_ctrl* communicates via parametrized signals through channels with the instance *we1*:*WallElem* which communicates in the same way with *ht1*:*Heat* where the heat transmission equation is solved. *WallElem* has the task to integrate radiation, convection and conductance. This small example shows how the computational tasks are distributed to the processes of the different objects.

The processes are modeled as extended state transition graphs. Actions model functionality, for example the heat transfer calculation. Because of the distribution of the tasks, the process models are quite simple and can be reused to a large extent for the purpose of simulation. In total, the example results in 13 different processes. As mentioned in the previous chapter, all processes either calculate their own rate of calculations or are triggered by changes from other objects. In this paper we will not go into further details of process modeling.

Figure 3 also shows the communication channels as arcs. Only the process *WallSegm_ctrl* can communicate with external objects. All internal instances of *WallSegm* can only communicate with the process. This restriction was introduced for the purpose of a standard interface of all objects. The result is that channels only exist between objects with a direct *parts* relation, drawn as arcs in Figure 1. In the example, a *Room* can only communicate with an adjacent *WallSegm* by sending signals through *Floor*.

After all models have been entered in the editor of the SDT toolbox, the C-code generator and compiler can be started. This also generates a runtime environment for the scheduling of all processes. In the example in Figure 1, a process for each instance is created, 316 in total. It is therefore very important for the execution time that process scheduling is efficiently implemented.

A SIMULATOR EXAMPLE



ig.4: Floor plan of building 32, floor 4

A medium size realistic example of a university top floor as shown in Figure 4 with 27 offices and labs of different sizes and three connected hallway sections was used to show that the approach can be applied to building simulation. Two large labs are modeled as three rooms each. Wall types include heavy concrete ceilings, insulating brick outdoor walls, and lightweight plasterboard-mineral wool indoor walls. All 70 windows have Thermopane glass. The thirty doors are of a heavy soundproof type. Heating is provided by 70 hot water radiators with motor valve actuators. All rooms have electrical equipment as additional heat sources. Total floor area is about 1000 m^2.

The thermal environment is modeled by temperature sensors for the outdoors, the adjacent staircases, the floor beneath, and the radiator hot water supply. Solar radiation is modeled by six sensors measuring the ra-

diation perpendicular to the six outdoor wall directions. All outdoor values have been taken from our own recordings with a resolution of 2 minutes [http]. The four indoor temperatures have been set to constant values.

The action of persons was modeled by manually generated files with events for opening or closing windows or doors. The control system events to control the radiator valves were also inserted in this input file for the purpose of testing the simulator stand-alone.

As an interesting detail, a radiators is modeled as heat transport by water flow in a rectangular container with heavy walls. The container is divided into sections along the flow, resulting in a simple numerical solution. With this model temperature distribution and dynamic effects can be modeled more precisely than with simple energy loss models. Together with the time response of the motor driven valves, feedback control of the water flow can be dynamically tested.

As the first step in a controlled software engineering process, the component structure of the real building floor was mapped into an object type-instance graph. A root object was added for the communication with the environment and for the interactive simulator control through Java graphical user interfaces (GUI). This first step resulted in 34 object types. The instance count resulted in 904 objects. This relation shows a high degree of regularity.

The topological relations of the floor are represented by regular expressions in a second step. Relations between objects with direct part-of relations have been excluded, because due to the used SDL block type templates, direct signal channels exist. Topological relations exist between rooms and wall segments, but, for example, also between sensors and actuators and building components. Topological relations exist between individual instances. Therefore, unique instance names are constructed as explained in chapter “Simulator Object Structure”. By utilizing regularity and wildcard tokens in the expression, the necessary number of expressions is 17.

In the third step the object graph was translated into SDL block type diagrams by editing the template. This mechanical work could be automated. Due to the similarity of buildings, a high degree of reuse of models is possible in future projects.

The last constructive step was the creation of the processes. Processes define the behavior and the function of the objects and are described by extended state transition graphs. A typical object is the heat transfer through a layered wall element. States define the initialization and continuation phases of the simulation. All calculations are done at transitions between states. State transitions are either triggered by external signals, for example temperature changes in adjacent rooms, or by internal timers.

The calculations implement the numerical solutions for the physical models. Because of the simplicity of the objects, very simple solutions as for example the explicit method for the Fourier equation could be used. All numerical solutions have first been tested with Matlab, experimenting with different parameters for time and space resolution.

Timers are controlled at runtime to implement the dynamic time-step control. The upper limit for time intervals is set by the stability requirements of the numerical solution for the object’s simulation function. Since external signals lead to immediate calculations, the next time interval starts at this event. Delays to prevent change avalanches, as explained in the chapter before the previous, are considered by additional timers.

Another task of the processes is signal distribution according to the regular expressions that describe the topological relations. This is a straightforward task that could be automated.

In order to keep the number of different object types and thus process types small, object parameters are distributed before simulation starts. For the wall segment example, all dimensions, material properties, and change propagation limits are parameters. Parametrization could be extended to structural properties, for example the number and order of layers in a wall segment. This was not implemented because it would complicate the processes considerably. Instead, a library of wall structure types has been set up for reuse.

Reuse is used for all object types. By including optional components or functions in reusable types, reuse can be extended. Unnecessary components or function can simply be deleted or commented out during an adaptation step to make objects more specific. As an example, all indoor wall elements have doors and windows in the reuse library. Due to this kind of reuse the effort for the construction and testing of processes was high for the first simulator, but is much smaller for future projects.

After the process of constructing the SDL models, the complete simulator code and runtime environment is generated automatically by the SDT tool and compiled into an executable simulator. This simulator exactly models the building part of interest without redundancy and integrates only the functions of interest. Nevertheless, the generated C-code consisted of 500,000 lines of code for our 27-room example. By setting change propagation limits, speed and accuracy, especially dynamic accuracy, can be traded.

SIMULATION RESULTS

Simulation was conducted with synthetic test files and with 24 hour weather files as measured in Kaiserslautern during February 2000. The weather files were combined with activities of persons opening and closing doors and with assumed reactions of a control system, opening and closing radiator valves. These activities were used to test the dynamic reactions to sudden events, in the same way step functions are used to test linear devices in electrical engineering. Extreme cases as for example opening of all 60 windows at the same time were used to test for real-time limitations. In the following chapter we will present some interesting samples of in- and outputs.

As a first result simulated room temperatures of three rooms in reaction to different constructed events are shown in Figure 5. At time 0s the outdoor temperature drops from 15 to -20°C and goes up to 20°C at 2000s. All rooms react in a reasonable way. The radiator valve in rm5 starts to open at 100s and is fully open at 500s. The nominal radiator power is 1000W at 20°C room temperature. A slow increase in room temperature can be observed. More dramatic is the reaction to opening the window in rm3 100% at 300s, closing to 20% at 900s and fully at 2000s. At 1000s additional solar radiation of 100W enters all rooms. Rooms rm4 and rm5 show a small increase in temperature.

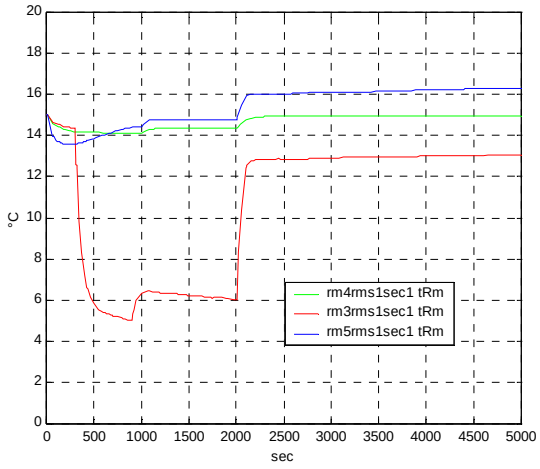


Fig.5: Simulated temperatures in rooms rm3 (bottom), rm4 (center), and rm5 (top) in group rms1 of section sec1

The experiment cannot prove the correctness of the simulation. Proofs of correctness are very difficult in building simulation. Our results so far can only show results as they would be expected. Even if we had measured room temperature values, we would not have enough precise data for the building construction, for example the air leakage of windows and doors. But the test demonstrates the expected dynamic behavior. For most control applications this is sufficient. Control algorithms have to be robust against parameter variations.

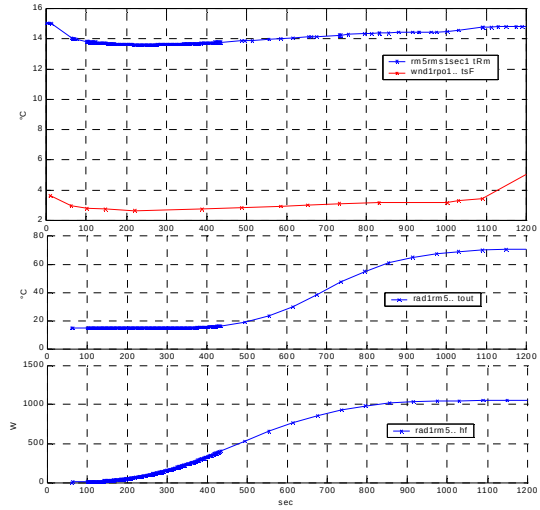


Fig.6: Radiator behavior for valve opening.
From the top: Room, window indoor surface, and hot water return temperature, radiator heat gain

The next test looks at the dynamic behavior of a radiator in more detail. Figure 6 shows three temperatures and the heat gain. The delayed increase in the return temperature and the time constant of the rise with measurements at radiators at our floor. The marks show when the object performed a calculation. After the valve starts to open water flow changes rapidly and the valve sends signals every two seconds to the radiator. After the valve is open, calculation intervals drop to an average of 60s. This is a nice example of dynamic time step control.

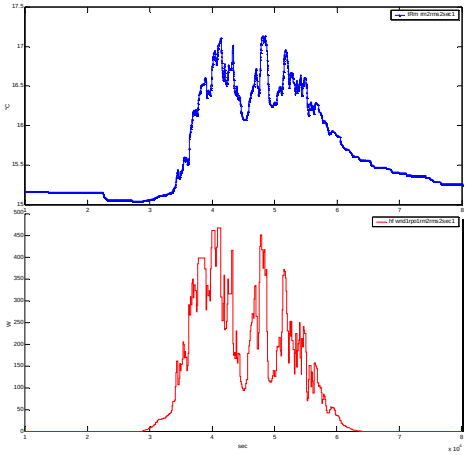


Fig.7: Simulated room temperature (top) and heat flow through the room windows (bottom) as reaction to measured horizontal solar radiation load

As a test with real environment data Figure 7 shows the influence of solar radiation on the room temperature. The strong fluctuations result from clouds passing the sun. The windows are facing SW. The heat

flow starts at 8:30 and ends at 17:00. The room temperature decrease is delayed because of the heavy concrete floor and ceiling. The shown examples may be sufficient to give confidence into the dynamic behavior of the simulator.

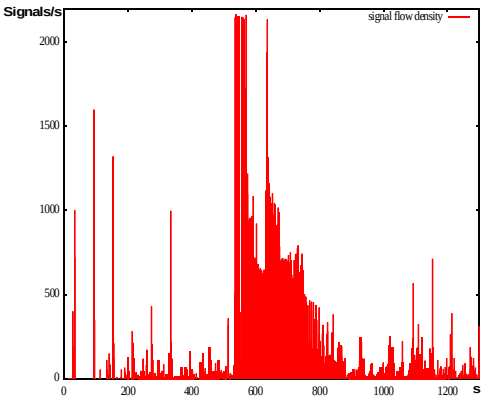


Fig.8: Signal frequency during real-time simulation of worst case scenario (the time is about 70s ahead of real-time)

With a different set of tests we examined the real-time and accelerated real-time features. As a worst case scenario all windows and all radiator valves were opened at time 500s. This, naturally, results in a burst of change signals. At 600s the outdoor temperature drops from 20 to -20°C, oscillates twice at 1s intervals and stays constant at 0°C until at 1000s temperature resumes measured outdoor values at about 8°C. Figure 8 shows that the measured signal frequency reaches saturation at 2200 signals per second for a short time. Since every signal causes a process switch and the signal monitoring causes some overhead, this is a reasonable figure for the simulation platform HP-J5000 with 440MHz clock.

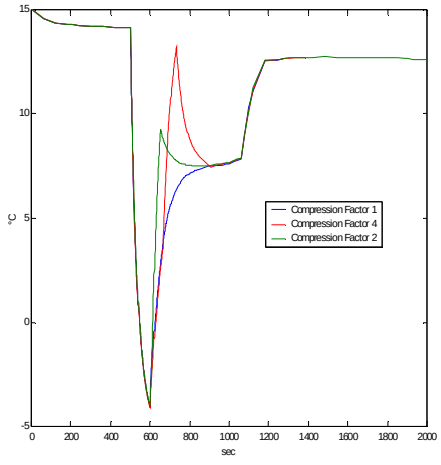


Fig.9: Simulated room temperature at different real-time compression factors

Saturation means that signal reaction may be delayed or even suppressed. We therefore checked the simulator output for a room temperature at different accellerations. At factor 1 the output reacts normally as we assume. At factors 2 and 4 the drop at 500s cannot be distinguished from the one with factor 1, but the rise is delayed, causing overshooting. Still, at 900s the behavior returns to normal. We call this behavior fault tolerant with graceful degradation of accuracy.

Further tests have shown that with this example of about 1000 building components modeled as independent processes and with “normal” environment change frequencies acceleration factors between 10 and 100 could be reached without degradation of dynamic behavior.

CONCLUSION

We have shown that building simulation based on communicating objects is a viable alternative to equation solver based approaches. It has been shown that the dynamical behavior correlates to the actual behavior of buildings. We plan to compare our results to those of other simulators as it was done by other authors [Mahdavi et al., 96].

Advantages of our approach are clearly the adaptive time resolution, resulting in small dynamic errors, the fast reactions to events, and the simplicity to extend and modify the functionality of the simulator and the possibility to integrate simulation and control. This simplicity is based on the fact that the simulator is modeled at a very abstract object based level and code is generated automatically. High reuse potentials further simplify the modeling process.

REFERENCES

Mahdavi, A.: “SEMPER: A New Computational Environment for Simulation-based Building Design Assistance”, Proceedings of the 1996 International Symposium of CIB W67 (Energy and Mass Flows in the Life Cycle of Buildings). Vienna, Austria, 1996.

“Dymola: Dynamic Modeling Laboratory”, Dynasim AB, Sweden, (http://www.dynasim.se/)

Diehl, A. and L. Litz: ”Object-Oriented Top-Down Modeling for Dynamic Simulation of Compression Plants”, Proc. of the 20th International Congress of Refrigeration, IIR/IIF, Sydney,1999.

Schütze, M., J. P. Riegel, and G. Zimmermann: “PSiGene - A Pattern-Based Component Generator for Building Simulation”, Journal Theory and Practice of Object Systems (TAPOS), Vol. 5, No. 2, 1999.

Olsen, A. et al.:”Systems Engineering Using SDL-92”, Elsevier Science B.V., 1994.

Telelogic AB, (http://www.telelogic.com)

(http://hauspc1.informatik.uni-kl.de/datalogs/)

Mahdavi, A., S. Lee, R. Brahme, S. Kumar, P. Mathew, and V. Pal. “Computational Evaluation of Five Energy Simulation Programs in View of Residential Applications,” Report for the Susquehanna Project, Center for Building Performance and Diagnostics, Carnegie Mellon University, Pittsburgh, PA, May 23, 1996.