EXTENDING BUILDINGS' THERMAL-DYNAMICS MODEL IN MODELICA WITH EXTERNAL ILLUMINATION MODEL

A. Sodja, B. Zupančič
Faculty of Electrical Engineering, University of Ljubljana

Corresponding author: Anton Sodja
Faculty of Electrical Engineering
University of Ljubljana – 1000 Ljubljana, Tržaška 25 – Slovenia
anton.sodja@fe.uni-lj.si

Abstract. An intelligent dynamic building's envelope is a promising concept for future low-energy building design. For assuring pleasant living conditions multiple quantities have to be controlled and due to complexity and diversity of systems (living and working rooms), the corresponding control can become very complicated. A good reusable model can play crucial role in the successful control design procedure.

This paper presents integration of the Radiance lighting simulation tool in our previously built model of the building's thermal dynamics in Modelica. With incorporation of the Radiance in Modelica a calculation of illumination and thus fully functional model is obtained, which is due to advantages of object-oriented modeling easily reusable and extensible to various types of buildings.

1 Introduction

The concept of modern energy-efficient building incorporates use of solar energy to reduce the requirement for active heating and cooling sources in assuring thermal comfort. There are many ideas for the design of energy-efficient buildings [9]. An important aspect of the design is the harmonization of the solar energy transferred through the glazing with the illumination of the interior [11, 5]. A shading is needed to prevent excess glittering occurring especially during low winter sun.

One approach to achieve the harmonization of proper illumination and solar energy gain is automatic control of window shading [3, 8] by a suitable controller. However, at the same time minimization of the cost of active heating and cooling respectively, should be achieved. The active system of shading or, more generally, dynamically-changing building's envelope, provides a good possibility to improve low-energy buildings' design and simultaneously assure pleasant living condition along with considerable energy savings. On the other hand, the required control algorithms tend to be very complicated due to the complexity and multivariability of the controlled systems (at least the illumination and temperature of the interior air must be taken into account). An efficient control design must be based on a model which incorporates thermal and lighting effects. With modeling and simulation before real implementation much greater flexibility is achieved and development time is considerably shorten.

Our previous work was based on experimentations with a real test chamber [6]. To assist the control development of the illumination and thermal status and to improve knowledge about the system (test chamber), we started with modeling in *Matlab/Simulink* [12]. This model was found to be inappropriate due to disadvantages of the non-object-oriented modeling tool when the extension to experimentations on a real-world buildings are required. So we decided to reimplement it in a more powerful modeling tool *Dymola* [10] which offers modeling in object-oriented modeling language *Modelica* [2]. Thermal dynamics of the building in our model is described with differential-algebraic equations using energy balance principle. With object-oriented approach an intuitive, reusable and easily expandable modeling of buildings' thermal dynamics is enabled. Similarly flexible and easily maintainable model was desired also for illumination. For this purpose we adapted *Radiance lighting simulation and rendering tool* [7] and incorporated it in *Modelica* model of thermal dynamics.

In the work *Radiance* illumination model is presented together with the developed interface to *Modelica*. The usability of the proposed approach is later illustrated on an example of the test chamber illumination.

2 Illumination model in *Radiance*

Radiance lighting simulation tool is a powerful rendering package using backward ray-tracing and is primarily intended for visualization and lighting analysis [7]. The tool reads scene-description sources and then generates image of a specific view or trace a single ray from a particular stand-point in a selected direction. During the simulation run of the building's thermal response and illumination also the scene changes. In particular, the sun's position (sun inclination angle) and geometry of the transparent parts of the building's envelope are calculated. In order to diminish overhead of reading scene-description sources anew in each simulation step, the *Radiance* tracing utility was changed to re-read only parts of the scene that changes during the simulation run. That implies separating scene-description sources for static and dynamic part of the scene. In contrast to static-scene description, in the dynamic part some dimensions/quantities (e.g., shape of windows' shading, intensity of solar radiation, etc.)

of the scene's objects are designated as variables (beside initial value, also the name of the variable is provided). So, *Modelica* model sends two types of information to *Radiance* submodel: the change of scene (change of variables' values) and position and orientation of the spot of interest. Obviously, *Radiance* subsystem sends back to *Modelica* the information about illumination value of the spot.

The interface at the side of the *Radiance* tracing utility consists of four functions (the *Radiance* API):

```
int  init_radiance (char *octree , char *dyn_files );

int  finalize_radiance ();

void  set_variable (char *varname, double val );

int  get_illumin (double *orig , double *direc , double * retval );
```

The first two represent initialization and finalization routines. In initialization one the scene data is read, namely the static part from the file `octree` and dynamic part from the files stated as comma-separated-list `dyn_files`. The third function sets variable `varname` to value `val` and the model is not refreshed (the dynamic part of the scene is re-read) till the forth function, in which the majority of the work is done, is called. Function `get_illumin` receives position and orientation (`orig` and `direc` respectively) of the spot of interest as the input parameters and returns computed illumination value in the variable `retval` (at the destination to which `retval` points to). In the finalization routine, memory allocated in initialization phase is freed.

The modified *Radiance* tracing utility is compiled and two static libraries are generated: *librtrace.a* and *librtrad.a*. The declaration of API (functions presented in this section) is made available in the header file *radiance.h*. The header file and libraries are then used in the process of translation and compilation of the *Modelica* model as illustrated in Figure 1.

3 Modelica Interface

Models in *Modelica* can interact with other models through the use of external functions [4, 2] written in other programming language. *Dymola* in current version supports only *C* language [1]. For each external function a Modelica interface must be declared.

In our example, the *Radiance* API consists of four functions (described in the previous section), two of them, initialization and finalization routines, should be called at the beginning and at the end of the simulation run respectively and the remaining two are called during the simulation run. In object-oriented manner, we organize the access to these four functions as a class. Each function have a Modelica interface nested inside the class and initialization and finalization routines of the *Radiance* tracing utility are called at initialization and termination of the class instance (that means before and after simulation run). In Figure 2 the principle of the communication

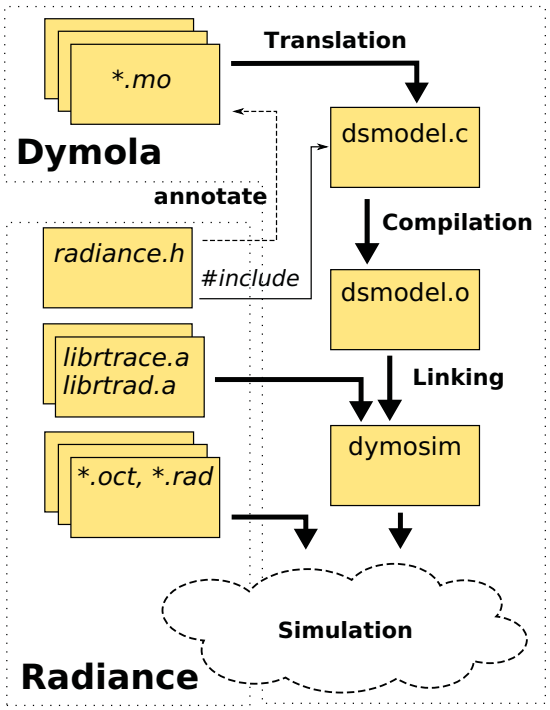

Figure 1: Scheme of translation and compilation of the *Modelica* model with *Radiance* extension.

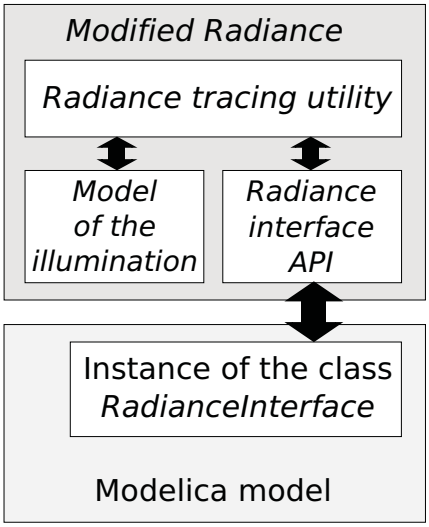

Figure 2: Scheme of communication interfaces of *Radiance* tracing utility and model in *Modelica*

between *Radiance* and *Modelica* model is depicted: the communication interface on the side of the *Radiance* provides four *C* functions which are called inside our interface class in *Modelica*. The information is finally conveyed to other parts of the model by use of *Modelica* connectors. The listing of the obtained interface class is:

```
partial block RadianceInterface " Interface to Radiance"
  parameter String octree "path to the octtree with static data";
  parameter String dynfiles "comma separated list of files ";
  annotation(Include="#include <radiance.h>", Library={"rtrace","rtrad"});
protected                                                                    5
  function initRadiance " Initialize connection with Radiance"
    input String octree;
    input String dynfiles;
    output Integer status;
    external "C" status = init_radiance(octree, dynfiles);                  10
  end initRadiance;

  function termRadiance "Terminate connection with Radiance"
    output Integer status;
    external "C" status = finalize_radiance();                              15
  end termRadiance;

  function setVariable
    input String variable;
    input Real newValue;                                                    20
    external "C" set_variable(variable, newValue);
  end setVariable;

  function computeIllumination
    input Real[3] origVec;                                                  25
    input Real[3] direcVec;
    output Real illumination;
    external "C" get_illumin(origVec, dirVec, illumination);
  end computeIllumination;
                                                                            30
  final parameter Integer status = initRadiance(octree, dynfiles);

equation
  when initial() then
    assert(status == 0, "Radiance intitialization error");                  35
end when;
  when terminal() then
    assert(termRadiance() == 0, "Radiance termination error");
  end when;
end RadianceInterface;                                                      40
```

It can be observed that *RadianceInterface* is declared as `partial block`. The `block` is a special restricted class type in *Modelica* which have no internal state and `partial` designates that an instance of the class is not allowed to be created. That is because the class represents solely a *Modelica* interface class to *Radiance* models. A complete class suitable for inclusion in our model of building's thermal dynamics must define input/output connectors and is thus limited to only a single *Radiance* scene configuration. In object-oriented language such as *Modelica* the general-purpose parts of the model can be elegantly separated from a more specific ones through the use of inheritance. Such approach is used also in this case. This class is intended to be a base class for interface classes created for specific *Radiance* illumination models.

Further the needed parameters are defined as arguments for initialization function and in line 4 the instructions to Dymola's compiler and linker in form of an annotation are given. Here definition of external *C* functions and libraries, which should be additionally included in the generation of model's executable, are listed.

Figure 3: View of the test chamber from the south side

At the end, initialization and finalization of *Radiance* subsystem must be invoked. For this purpose, *Modelica* offers two events: functions *initial()* and *terminal()* return true at time zero and at final time of the simulation run respectively. Events are handled inside `when` clauses in lines 34 and 37. However, initialization function of *Radiance* subsystem is not called during *initial()* event handling, but this event happens at time zero of the simulation run, when all initial values of the states are already determined. In the determination of initial states of the model, the simulator might claim output values from *Radiance* subsystem and in this case *Radiance* would crash while *Radiance* model is not initialized yet. So we introduce parameter *status* in line 43. It has a value returned by initialization function *initRadiance*. Correctness of the initialization (value of parameter *status*) is checked at the *initial()* event in line 31.

4 Illumination model of the test chamber

The used test chamber is shown in Figure 3. It is square-shaped with a single south-facing window. A roller blind with variable position is installed on outer side of the window. An illumination sensor is placed in the middle of the ceiling facing downwards (Figure 4).

The *Radiance* model of the test chamber is simple, consisting of only three objects that changes during the simulation run: position of the sun, intensity of the solar radiation and position of the roller blind (transparent area of the window).
The input data to *Radiance* model are the time of the day (needed to calculate position of the sun), solar radiation intensity (it is used also for estimation of the brightness of the sky) and the position of the blind (0 is completely opened and 1 is completely closed). The model is extended from the *RadianceInterface* model. In the extended model the time of the day is calculated as the sum of offset (the start time of the simulation run) and the simulation time which is available as a global variable `time` in *Modelica*. So only two inputs are needed: the intensity of solar radiation and the position of the blind while the output is illumination calculated from the position and orientation of the sensor on the ceiling. The inputs and outputs are defined as follows:

```
import SI = Modelica.SIunits;
import Modelica.Blocks.Interfaces.*;
RealInput SolarIntensity(redeclare type SignalType = SI.RadiantIntensity);
RealInput BlindPosition(redeclare type SignalType = Real (min=0, max=1));
RealOutput RoomsIllumination(redeclare type SignalType = SI.Illuminance);
```

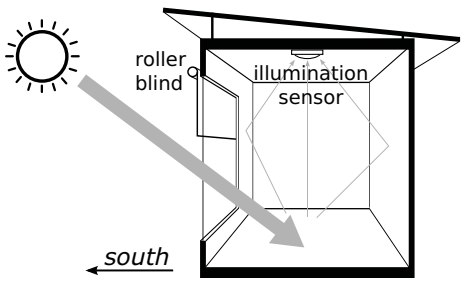

Figure 4: The scheme of test chamber with roller blind and illumination sensor configuration

Additional to input and output declaration, the following parameters have to be defined:

```
parameter SI.Time initTime;
parameter SI.Distance sensorPos[3];
parameter Real sensorOrient[3];
```

The first represents the start time of the simulation in seconds from the start of the year, and the last two are position and orientation of the sensor.

As computationally-expensive illumination computing in each simulation step should be avoided, the calculations only in discrete times are enabled. In our case 5 min interval is selected (the same as the sampling interval of the solar radiation-intensity measurements which are routed to the input in our model). At each sampling time the following calculation is performed:

```
when sample(0,300) then
  (hour,minutes) := time_indices(initTime+time);
   setVariable("hour",hour);
   setVariable("min",minutes);
   setVariable("blind_pos", BlindPosition);
  RoomsIllumination := computeIllumination(sensorPos, sensorOrient);
end when;
```

Firstly, the new dimensions' values are set with `setVariable` calls and then the new illumination value is recomputed by calling `computeIllumination`.

5 Conclusion

When modeling complex systems, like in the paper presented model of living conditions in buildings, a need to include submodels implemented in other modeling tools often arises. It can happen because the primary modeling tool is not powerful enough or a required submodel is already realized in an other modelling tool and reimplementation would mean an extensive work.

Modelica has a well defined interface for the use of external functions and we were able to implement clear interface to *Radiance* lighting simulation tool in the sense of easy integration to existing model of building's thermal dynamics. No nasty hacks were needed and easy reusability and extensibility, as one of the main advantages of the object-oriented modeling, were not lost or diminished.

However, a condition for the described smooth incorporation of external model in the *Modelica* is, that external submodel has no dynamics, i.e., it has no internal continuous states [4].

Additional issue which was not exposed in the paper is the requirement that interface class to *Radiance* must be a singleton. This is not assured with implementation in *Modelica* and also the language *Modelica* does not offer any feature which could be used to assure creation of only one instance of the class. The problem could be solved at the side of the *Radiance* by starting *Radiance* as a stand-alone process and establish connection to each process (e.g., through pipe or sockets). Of course it results in additional communication costs and more complex interface or in modifying *Radiance* program to support serval submodels simultaneously. However, that is very tedious task. Because we do not expect the mentioned situations in the future, we left this problem unsolved.

As the stress in the present work is on the interface development from *Radiance lighting simulation tool* to *Modelica*, the model of building's thermal dynamics with added illumination modeling is neither given nor validated. The only important fact is that the proposed modeling environments can successfully communicate.

In the future work, we intend to validate our model on a real building and use it for control design as well as for teaching.

6 References

[1] Dynasim AB. *Dymola User's Guide, Version 5.3d*. Sweden, Lund, 2004.

[2] Modelica Association. *Modelica - A Unified Object-Oriented Language for Physical System Modeling. Language Specification, Version 2.2*. February 2005. `http://www.modelica.org`.

[3] A. K. Athienitis and A. Tzempelikos. A methodology for simulation of daylight room illuminance distribution and light dimming for a room with a controlled shading device. *Solar Energy*, 72, 2002.

[4] T. Blochwitz, G. Kurzbach, and T. Neidhold. An external model interface for modelica. In *Proceedings of Modelica' 2008*, Bielefeld, Germany, 2008. The Modelica Association and University of Applied Sciences Bielefeld.

[5] E. L. Krüger and P. H. T. Zannin. Acoustic, thermal and luminous comfort in classrooms. *Building and Environment*, 39(9), 2004.

[6] M. T. Lah, B. Zupančič, J. Peternelj, and A. Krainer. Daylight illuminance control with fuzzy logic. *Solar energy*, 80, 2006.

[7] G. W. Larson and R. Shakespeare. *Rendering with Radiance*. Morgan Kaufmann Publishers, San Francisco, California, 1998.

[8] S. Onaygil and Ö. Güler. Determination of the energy saving by daylight responsive lighting control systems with an example from istanbul. *Building and Environment*, 38(7), 2003.

[9] I. Sartori and A. G. Hestnes. Energy use in the life cycle of conventional and low-energy buildings: A review article. *Energy and Buildings*, 39(3), 2007.

[10] A. Sodja and B. Zupančič. Some aspects of thermal and radiation flows modelling in buildings using modelica. In *Tenth International Conference on Modeling and Simulation*. EUROSIM-UKSIM, 2008.

[11] A. Tzempelikos. The impact of venetian blind geometry and tilt angle on view, direct light transmission and interior illuminance. *Solar Energy*, 82, 200.
[12] I. Škrjanc, B. Zupančič, B. Furlan, and A. Krainer. Theoretical and experimental fuzzy modelling of building thermal dynamic response. *Building and environment*, 36(9), 2001.