A Neutral Format for Building Simulation Models

Per Sahlin, Research Secretary
Swedish Institute of Applied Mathematics
Box 26 300, 100 41 STOCKHOLM, SWEDEN
and
Department of Building Services Engineering
Royal Institute of Technology, SWEDEN

Edward F. Sowell, Professor
Department of Computer Science
California State University Fullerton
Fullerton, CA 926 34
and
Department of Building Services Engineering
Royal Institute of Technology, SWEDEN

ABSTRACT

Much research has been directed towards development of software environments that allow easy construction of building simulation models of widely varying structure and purpose. For example, TRNSYS has been in use for a number of years. Recently, several new such environments have been proposed. In spite of a considerable variation in model description formats among environments, the underlying mathematical models of physical processes are very similar. While one of the principal aims has been to allow easy sharing of models between users of the same environment, it has not been clear how portability was to be provided between different environments. Another objective has been ease of component model definition, in order to encourage modifications and additions to model libraries. This paper addresses both of these issues, by proposing a neutral and natural format for component model expression. The proposed format encourages equation based model definition because such models can be converted to efficient algorithmic form if needed, whereas the converse is not always true. Nonetheless, algorithmic component descriptions are also supported in order to allow reuse of existing models. Other key features of the proposed format are typing and declaration of linkage elements between models, which allow development of compatible component families, and enhance model exchange and reuse. The proposal considers underlying

1. INTRODUCTION

There are currently several modular programs in use for simulation of buildings and associated service systems, e.g TRNSYS, HVACSIM+ and NEPTUNIX. Additionally, several new modeling environments for the same purpose have recently been proposed, and are in various stages of development [CLARKE 1985, SOWELL 1986, SAHLIN 1988]. All of these alternatives are similar in the sense that the mathematical models of components and subsystems are expressed in program modules that the user can interconnect as needed to define the wanted system model. The usefulness of any such environment depends on the availability of a library of predefined models for components in the intended application area, and on the existence of a simple mechanism for implementing new models when needed.

One might expect that component models could be interchanged among environments because, at a given level of idealization, the mathematical models of the physical processes are virtually the same. Unfortunately, this is not necessarily the case because each environment employs its own semantics and syntax for model expression and interconnection. Without some form of standardization of component model definitions the desired portability will be provided, at most, within modeling environments, but not between them.

This paper suggests a possible starting point for such a standard, namely a Neutral Model Format (NMF). The format is "neutral" in the sense that models are expressed in a general manner, rather than in the format of any existing or planned environment. The standardized definition encompasses only the essential information needed to express a model unambiguously. This information is formalized in order to allow automatic translation to the format of a particular simulation environment. The format is "natural," meaning that the definition employs terms and constructs as close as possible to the experience and training of scientists and modelers.

The focus, in this initial work, is on models with a basically continuous behavior. This includes building envelope as well as HVAC system models. Excluded are components such as thermostats with a dead band (hysteresis) and micro processor based controllers, which are better described in discrete time. Furthermore, the emphasis is on the machine readable mathematical description of the models. Systematic model documentation has been treated elsewhere [CLARKE 1984, DUBOIS 1988].

The discussion that follows begins with an overview of structured modeling principles that motivate the NMF. Many of these are inspired by the work of [ELMQVIST 1986] and [MATTSSON 1988]. This is followed by a description of the format, supported by small examples.

2. MODEL STRUCTURING PRINCIPLES

The NMF is based on a few principles that ensure generality:

1. Continuous models are expressed in terms of equations.

2. Variables and interconnections are typed.

3. Large models must allow hierarchical decomposition.

4. Validation is integrated into the modeling process.

The principles are briefly described and defended below.

2.1 Equation Modeling

The internal behavior of a continuous component of the NMF is described by a differential–algebraic system of equations which for the general case can be written

$$f(x, \dot{x}, p) = 0,$$

where f is a vector function of the variable vector x, its time derivative $\dot{x}$, and a parameter vector p. In all cases of interest here, this system of equations will be underdeter

mined; some of the x's will have to be given as functions of time.

Let us, for the sake of the discussion, separate between the *equation model* of a component and a *problem* for the same component, where the problem is the underdetermined equation model *together* with a selection of given variables. For example the equation model of a thermal resistance may be written

$$0 = q - UA(t1 - t2),$$

where q is the heat flow through the resistance and t1 and t2 are the terminal temperatures. Now, for this simple one-equation model three different problems, i.e. combinations of given and calculated variables may be posed:

1. t1 and t2 given and q calculated
2. t1 and q given and t2 calculated
3. t2 and q given and t1 calculated

All three problems are *well posed.* In the following, well posed will be used in the sense: able to produce a locally unique non–trivial solution. For more complex models only some selections of given variables will yield well posed problems.

Each component model in most current simulation environments, e.g. TRNSYS and HVACSIM+, is described as an equation model *along with a single input–output selection* (a problem in our sense). The component modeler makes this selection when the model type routine is written.

The pre–selection of given variables leads in some cases to limitations in the actual use of the models. Frequently a system modeler, using available types, would like to connect the inputs of one component with the inputs of another and similarly for the outputs. This, of course, is impossible and one of the component models has to be rewritten, with a different input–output selection. The system modeler is forced to become a component modeler and write, debug and compile Fortran code.

These difficulties are overcome in some of the more recently proposed environments (e.g. SPANK and Ida, formerly MODSIM [SAHLIN 1988]) by leaving the input–output designation to the environment. This will substantially increase the versatility of each component model.

The automatic input–output designation in more recent environments is done by keeping equation models separate from input–output selections until the components are actually connected together. This separation is only possible if equations are declared separately, the way they are in the NMF.

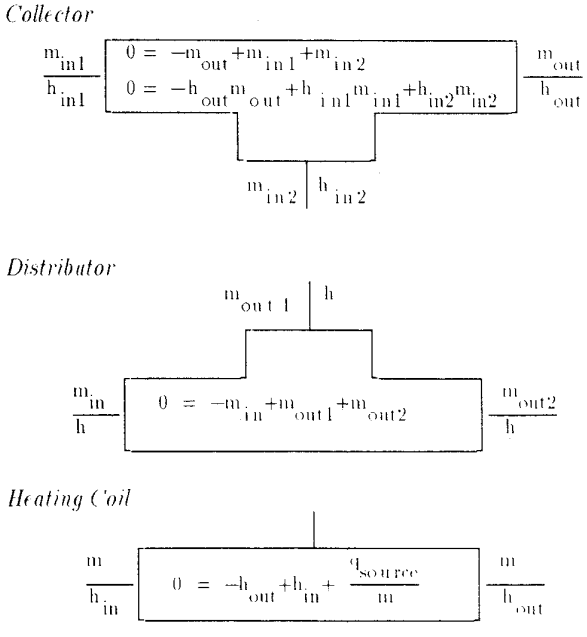
Since some environments can do without explicitly stated input–output designations in their component model format, one could argue that this information is redundant in the NMF, which should be free of environment specific non–essential information. There are however several reasons for including *one possible input–output designation* (one problem) for each NMF component model. Firstly, if this information was to be left out, automatic translation would be impossible for input–output oriented environments. Secondly, a viable input–output set is a part of the required validation procedure. That is, a component modeler has to demonstrate at least one well posed problem for a model.

2.2 Component Interconnection

Having focused briefly on the internal behavior of component models we turn to the interconnection mechanism between them. Little attention has been devoted to this topic in many of the past discussions on the development of common component libraries, although model reuse and exchange have been the primary motivations. However, one should be aware that sets of components developed by various groups will remain to be incompatible, even when stored in a common library, unless a structured way of constructing inter–component links is imposed. Otherwise, the sockets and the prongs simply will not fit together.

The development of a set of component models for a simulation task involves numerous decisions, some of which are crucial and others which are less fundamental in nature. Unfortunately, all of these decisions, not just the crucial ones, will later on influence the compatibility with other models. It is our aim here to provide a component format which encourages compatible choices among the trivial decisions without imposing any restrictions on the fundamental ones.

One of the initial crucial decisions to be made is the choice of a set of variables that will represent the behavior of the simulated system to an appropriate degree of accuracy. For example, in a simple HVAC circuit without cooling it might be sufficient to choose dry air mass flow rate and air enthalpy as the main variables carrying information between individual components. We are referring here to the set of variables involved in the *interaction* between components; additional variables may be used internally. Once this choice of interaction variables has been done, a *compatible family* of components can be developed. For the HVAC circuit this might involve e.g. a collector, a distributor, a heating coil and a simple zone model as shown in Figure 1.



Zone

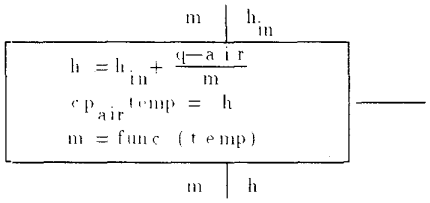


Figure 1.

The zone model has a control function, func, built in which determines the supply air flow rate as a function of zone temperature.

The choice of interaction variables is affected by the component model complexity. For example – if we wish to consider now moist air problems – a cooling coil model should include the effect of condensation, and thus some measure of air humidity must be included in the air characterization. If the cooling coil is to be used in conjunction with the previous models, the list of interaction variables to be carried around the circuit must be expanded. The principle here is that the component in need of the most information determines the interaction set of variables. Components with smaller needs will ignore unnecessary circuit variables.

Now, let us look at some of the trivial decisions for our sample case. Although the simulation in principle can be carried out using vastly different sets of units in each component, compatibility is enhanced if common units are used. This is an area where encouragement, via access to existing models, and mild punishment, via compulsion to write additional declarations, are likely to stimulate uniformity. For the sample HVAC circuit, a similar argument can be made concerning the choice between temperature and enthalpy as an interaction variable.

A mechanism for increased compatibility in this sense is *variable typing*. All variable types to be used in component models are declared globally in the NMF. A modeler who is about to introduce a new model in the library will use already declared types whenever possible.

The next step is to declare the groups of variable types that characterize compatible families. Such a group is called a *link type* in the NMF. Mass flow rate and enthalpy together and in this order is an example of such a type. Examples of the typing syntax are located in the beginning of section 3.

The link concept also allows an environment user to connect submodels at the interface level rather than variable by variable. This means, for example, that a fan outlet is connected with a cooling coil inlet as far as the user is concerned; in the background however several variables may be involved in the connection. Most current simulation environments, e.g. TRNSYS or HVACSIM+, operate on the variable level, so the link concept would in this respect simply be ignored for these. The more important library structuring effect of link typing is still retained.

In link supporting environments link types can be used to check whether a user is making meaningful connections. There are cases, however, when a strictly imposed typing concept is too restrictive. Controllers, for example, should be allowed to interface with various types of links. This is dealt with in the NMF by providing a generic link type which can contain any number of any type of variables. An environment can then check the individual variables in the connecting links for matching types rather than the links themselves. A generic variable type is also provided in order to allow for suppressed type checking on the variable level as well. [MATTSSON 1988] discusses ways of constructing more elaborate type checking in a modeling environment.

2.3 Hierarchical Decomposition.

Another fundamental concept for structured modeling is hierarchical submodel decomposition, i.e. one submodel within another in multiple levels. A composite building model could then, for example, be composed of several submodels, each one representing a floor. A floor is in turn built up of several zone models, which are built from wall models, and so on. One major advantage of this method is that it enables incremental modeling and validation. A modeler can make sure that e.g. a wall model behaves properly before it is used as part of a zone model, which then is similarly validated and so on, incrementally approaching the building level. Another advantage is that good graphical interfaces can be constructed for a corresponding hierarchical presentation of a model, where a user first gets an overall view of the system and then can zoom down for successively increased levels of detail.

Although most component models in the NMF will be used as part of composite or macro models on the environment level, the formatting of composite models themselves, i.e. interconnection templates, is not encompassed by the present proposal. There are several reasons for this, including the observation that small models are inherently more readily reused. At the same time we recognize that the process of component model development, within the NMF, could benefit from hierarchical decomposition, so eventual extension to this capability is an open issue.

2.4 Model Validation

Component model validation – in the sense of making sure that a model to some degree of accuracy reproduces the behavior of the actual physical device – can of course only be done if one has access to the device itself. There is no way to stop someone from using a library component in a non–intended way. The best thing one can do is to require extensive textual documentation to be provided along with the library entry, including the background of the underlying mathematical model. The documentation aspect of library building is beyond the scope of this paper.

The ambition of the NMF is to make sure that the entered models make sense from a mathematical perspective. Unfortunately, even this is quite a task. Existence of solutions to nonlinear equations is a very difficult subject and no general and practical theory exists. A model may work well over a particular parameter and variable range and be ill posed over another. In the end, we are left with the component modeler's ability to write robust models and to document them properly, including their ranges of validity.

What the NMF asks of a modeler is that a single problem – one input–output designation along with an equation model – is provided, and that its range of well posedness is specified. The well posedness range is specified in two different ways: firstly, in terms of explicit limits on the involved parameters and variables and, secondly, in terms of accompanying documentation. Responsibility for the existence of solutions for other possible input–output designations must be left to the targeted environments.

A number of methods of varying degrees of reliability are available to the component modeler in order to make sure that the single problem is well posed. These methods can be applied separately or jointly. Eventually, there may be software tools to assist in this process. The methods include:

1. *Functional Linear Independence.* This means that no model equation can be formed by a linear combination of the others.

2. *Matching.* There must exist a one to one matching between equations and variables in the designated output set.

3. *Regular matrix pencil.* For a general component model (from section 2.1) call the vector of the designated input set u and the corresponding output vector x yielding $f(x, \dot{x}, u, p)$, where $\dim(f) = \dim(x)$. Then the matrix (pencil):

$$\lambda\frac{\partial f}{\partial \dot{x}} - \frac{\partial f}{\partial x},$$

   where λ is some scalar, must be non–singular for all but a finite number of λ's and this must, of course, be true for the entire parameter and variable working range of the component [SÖDERLIND 1988].

4. *Numerical Testing.* This is the most reliable test and also one of the more practical. The idea is that the modeler finds some algorithm for solving the designated problem, e.g. a direct iterative scheme or a general purpose differential–algebraic integrator such as DASSL [PETZOLD 1982] or even a simulation environment, and then, by numerical experimentation, finds the range of well posedness in parameter and variable space. As a minimum, it must be shown that a solution exists in the intended operational regime.

Some solvers take advantage of information about "undesirable inverses" of individual equations. The basic idea here is that a scalar equation, for example $h(x,y) = 0$, may be readily inverted to yield $x = g1(y)$ where g1 is a well behaved function, but the inverse $y = g2(x)$ may be problematical. One possible problem is that the function g2 may not be well behaved numerically. For example, $dg2/dx$ may become infinite in the range of interest, or for environments that develop the inverses symbolically, g2 may not be obtainable as a closed form expression, or even if obtainable it may have poor numerical properties or be unwieldy. The list of undesirable inverses is optional in the format and can be left out for the convenience of local modeling in environments that do not use this information.

3. THE FORMAT

In this section the basic elements of continuous NMF models are explained and exemplified. Some more advanced features have been omitted due to space limitations. A formal syntax description has been formulated but is not included in this paper.

3.1 Global declarations

As previously motivated variable types and groups of such types, link types, are declared globally. The global declarations are then referenced from each component model declaration. Parameter types and constants are also declared globally within a library of component models.

Some examples of global declarations are

```
VARIABLE_TYPES

/* name      unit          kind*/

temp         "Deg–C"       CROSS
heatflux     "kW"          THRU
massflow     "kg/h"        THRU
enthalpy     "kWh/kg"      CROSS

LINK_TYPES

/* name              variable types ... */

heat_flow            (temp, heatflux)
heat_source          (heatflux)
mass_enthalpy        (massflow, enthalpy)

PARAMETER_TYPES

/* name              unit */

heatflow             "kW"
heat_capacity        "kWh/(kg Deg–C)"
massflow             "kg/h"
temp                 "Deg–C"

CONSTANTS

/* name        value        unit */
stef_bolz      5.77E–11     "kW/(m2 K)"
```

The first two fields of a variable type declaration need no explanation, but "kind" may not be familiar. All variables can be categorized as being of either direction dependent flow–type (e.g. mass flow, heat flow, electrical current, torque and force) or direction independent potential type (e.g. temperature, pressure, enthalpy, voltage and position). The physics of flow–type variables says that they should sum to zero when two such variables are connected together. They are traditionally called through variables and will be called so here as well. Potential–type variables are on the other hand set equal to each other when connected. They are called cross variables.

A link type declaration is a named list of a set of variable types.

3.2 Continuous Model Elements

The elements of continuous models will be introduced incrementally, starting with the collector model of the sample HVAC circuit. All examples of this paper are designed primarily to illustrate the NMF. Most of them have not been tested in practical simulation.

```
CONTINUOUS_MODEL mh_collector

ABSTRACT "A thermal tee–piece model for bringing together two
separate streams of fluid"
```

```
EQUATIONS

/* mass balance */
0 = -m_out + m_in1 + m_in2    BAD_INVERSES () ;

/* energy balance */
0 = -h_out*m_out + m_in1*h_in1 + m_in2*h_in2
BAD_INVERSES (h_out, h_in1, h_in2)

LINKS

/*      type            name        variables .... */
mass_enthalpy           outlet      POS_OUT m_out, h_out ;
mass_enthalpy           inlet1      POS_IN m_in1, h_in1 ;
mass_enthalpy           inlet2      POS_IN m_in2, h_in2

VARIABLES

/*type        name    role def  min        max   descr.*/

massflow      m_out   OUT  0.   0.         BIG   "outlet massflow"
massflow      m_in1   IN   0.   0.         BIG   "inlet1 massflow"
massflow      m_in2   IN   0.   0.         BIG   "inlet2 massflow"
enthalpy      h_out   OUT  0.   -BIG       BIG   "outlet enthalpy"
enthalpy      h_in1   IN   0.   -BIG       BIG   "inlet1 enthalpy"
enthalpy      h_in2   IN   0.   -BIG       BIG   "inlet2 enthalpy"

END_MODEL
```

3.2.1 Equations

As previously motivated, the internal behavior of continuous components is described by a system of scalar equations, each of which is written

<expression> = <expression>,

where an expression may be a single variable, a first order time derivative, a parameter, a number, or some mathematical combination of the above. The aim is to keep the syntax as "natural" as possible. Expressions may also include references to separately defined functions written in a regular programming language. The order of the equations is completely arbitrary; the solution procedure is beyond the scope of the component model.

The optional list of bad inverses is associated with each equation.

Variables may be arrays, and vector operations can be defined through external subroutine calls. The syntactical details of such operations have been omitted here.

3.2.2 Links

All variables that connect the model with neighboring models must appear in a link declaration. The link type must be either globally declared or GENERIC. Each THRU variable in the link is specified in terms of its direction of definition.

3.2.3 Variables

Each continuous model variable is declared in seven aspects:

1. Type. Each type that is referred to must be either globally declared or of the GENERIC kind.

2. Identifier. For array–type variables, index ranges are given.

3. Role. As mentioned earlier, one feasible *problem* is specified for each model. Variables are cast to play a certain role in this problem as either given (IN) or as calculated (OUT).

4. Default value. Most environments will provide defaults for initial values (of state variables) and of initial value guesses (for algebraic variables).

5.& 6. Minimum and maximum limit of the allowed range. Each variable is given a range, within which the model is valid.

7. Explanatory text string.

Variables that only appear in the links (interfaces) of a model – i.e. which do not appear in any of the equations – are declared in the same way. Role is irrelevant for these variables, but is conventionally set to be IN.

The mh_zone model involves some additional complexity:

```
CONTINUOUS_MODEL mh_simple_zone

ABSTRACT "A zone model with built in control of supply air flow
rate as an external function of zone temperature"

EQUATIONS

/* zone energy balance */
h = h_in + q_air/m      BAD_INVERSES () ;

/* temperature—enthalpy conversion */
cp_air*temp = h      BAD_INVERSES () ;

/* required supply air mass flow */
m = func (temp, m_max, m_min, t_max, t_min)
BAD_INVERSES (temp)

LINKS

/* type                name                variables .... */

mass_enthalpy          air_inlet           POS_IN m, h_in;
mass_enthalpy          air_outlet          POS_OUT m, h;
heat_flow              zone_air            temp, POS_IN q_air;

VARIABLES
/*type        name    role  def  min        max   descr.*/

enthalpy      h_in    IN    0.   -BIG       BIG   "entering air"
enthalpy      h       OUT   0.   -BIG       BIG   "zone air"
massflow      m       OUT   .01  SMALL     BIG   "supply air"
heatflux      q_air   IN    0.   -BIG       BIG   "air heat source"
temp          temp    OUT   0.   -BIG       BIG   "air temperature"

PARAMETERS

/* type               name        def     min     max   descr*/

heat_capacity         cp_air      28E-5   0.      BIG   "air heat
capacity"
massflow              m_max       50.     SMALL   BIG   "maximum
ventilation rate"
massflow              m_min       .01     SMALL   BIG   "minimum
ventilation rate"
temp                  t_max       25.     0.      BIG
"temperature at which supply air is set to minimum"
temp                  t_min       18.     0.      BIG
"temperature at which supply air is set to maximum"
```

```
FUNCTION

FLOAT  func (temp, mmax, mmin, tmax, tmin)

LANGUAGE  F77

INPUT
    FLOAT temp, mmax, mmin, tmax, tmin;

CODE

REAL FUNCTION FUNC(TEMP, MMAX, MMIN, TMAX, TMIN)

REAL TEMP, MMAX, MMIN, TMAX, TMIN

IF (TEMP.GE.TMAX) THEN
    FUNC = MMIN
ELSEIF (TEMP.LE.TMIN)
    FUNC = MMAX
ELSE
    FUNC = (MMIN - MMAX)*TEMP/(TMAX - TMIN)
ENDIF

RETURN
END

END_CODE
END_MODEL
```

3.2.4 Parameters and Model Parameters

Parameters are used to adapt a generally formulated equation model to the behavior of an actual device. They are declared under two separate headings: Model Parameters and Parameters. The former allow a user to adapt a model structurally, to provide internal flexibility in numbers. Their purpose and use are further explained in section 3.2.6. The latter class of parameters are the more straightforward, they specify behavioral properties such as size, heat capacity, thermal conductivity etc.

3.2.5 Functions

Separate functions or subroutines are used either in the equation model or for parameter processing, which is explained below. They may be written in various well known programming languages, thus making it possible to reuse already existing code within the format context.

3.2.6 Flexible Model Descriptions

The main objective of simulation environments, rather than simulation programs, is to provide increased modeling flexibility. This flexibility can be separated in *structural* and *behavioral* flexibility. The division between the two is somewhat diffuse; structural flexibility means that mathematical models of structurally differing physical systems may be built, and behavioral implies that a structurally fixed model may be adapted to simulate quantitatively different systems of the same basic structure.

The possibility to interconnect component models in various configurations provide structural flexibility in simulation environments, while in the more traditional programs the emphasis clearly is on behavioral flexibility.

Flexibility in numbers. It is convenient to provide some degree of structural flexibility *within* a primitive component model. For example, a wall model can in principle be constructed by connecting a number of separate instances of thermal resistance and mass models in series. The potential accuracy of the model is then determined by the number of layers (masses). However, this approach is cumbersome in several ways. The addition of a layer in order to alter the accuracy is a modeling operation to be carried out in several steps, e.g. instantiate a model, set its parameters and connect it. It would be much easier if this flexibility in numbers could be contained within a single wall model. The NMF has a construct for flexibility in numbers, the FOR statement. It is illustrated by a finite difference model of a wall.

```
CONTINUOUS_MODEL  thermal_wall

ABSTRACT "A 1D finite difference wall model"

EQUATIONS

/* space discretized heat equation */
FOR i = 2, n_layers -1
    c*t'[i] = t[i - 1] - 2.*t[i] + t[i + 1] ;

c*t'[1] = taa - 2.*t[1] + t_[2] ;
c*t'[n_layers] = t[n_layers - 1] - 2.*t[n_layers] + tbb ;

/* boundary conditions */
0 = -ta + .5*(taa + t[1]) ;
0 = -tb + .5*(t[n_layers] + tbb) ;
0 = -qa + d*(taa - t[1]) ;
0 = -qb + d*(tbb - t[n_layers])

LINKS

/*  type        name        variables .... */

heat_flow       a_side      ta, POS_IN qa;
heat_flow       b_side      tb, POS_IN qb;

VARIABLES

/*type    name                    role def min          max
                                                  description*/
temp    t[1..n_layers]          OUT  0.   -BIG    BIG
"temperature profile"
temp    ta                      OUT  0.   -BIG    BIG
"a-side surface temp"
temp    tb                      OUT  0.   -BIG    BIG
"b-side surface temp"
temp    taa                     OUT  0.   -BIG    BIG
"a-side virtual temp"
temp    tbb                     OUT  0.   -BIG    BIG
"b-side virtual temp"
heatflux    qa                  IN   0.   -BIG    BIG
"a-side entering heat"
heatflux    qb                  IN   0.   -BIG    BIG
"b-side entering heat"

MODEL_PARAMETERS

/*type name          min max      decription */

INT   n_layers      3  BIGINT    "number of layers"

PARAMETERS

/*type      name      def   min     max
                                  description */

c-type      c         10.   SMALL   BIG   "rho*cp*dx*dx/
(lambda*3600.)"
d-type      d         1.    SMALL   BIG   "lambda*a/dx"
```

```
/* easy access parameters */

area          a        5.    SMALL  BIG  "wall area"
length        thick    .25   SMALL  BIG  "wall total thickness"
heat_trans    lambda   .83   SMALL  BIG  "heat transfer coeff"
density       rho      2050. SMALL  BIG  "wall density"
heat_capacity cp       1.2   SMALL  BIG  "wall heat capacity"

PARAMETER_PROCESSING

wall_par( n_layers, c, d, a, thick, lambda, rho, cp)

FUNCTION

VOID wall_par (n, ccoeff, dkoeff, area, thick, lambda, rho, cp)

LANGUAGE F77

INPUT
    INT n;
    FLOAT area, thick, lambda, rho, cp;

OUTPUT
    FLOAT ccoeff, dcoeff;

CODE
/* The Fortran code is omitted*/
END_CODE
END_MODEL
```

A further example of the FOR statement comes from the mh_zone model. The version we have discussed so far has a single qT–link enabling it to be connected to one external component like a wall or a thermal resistance. It would be better to have a parameter within the model, a model parameter, which determines the number of available qT–links. Then a suitable number can be selected during system modeling and the zone core can be connected to an arbitrary number of walls.

Before we go into the feature provided for run time behavioral flexibility, the parameter processing header of the wall model deserves to be mentioned.

3.2.7 Parameter Processing

All the various named coefficients of the equation model are declared as parameters, but in addition to this, extra parameters may be declared in a model. Frequently, the mathematical characterization of a model, i.e. the parameters that appear in the equation model are quite different from those that an engineer spontaneously would choose to specify the corresponding physical device. For example, a zone model, which accounts for long wave radiation between surfaces, would have view factors (in some form) appearing in the equation model. However, a user of such a model would normally not prefer to specify these directly but rather the sizes, orientation, and reflectance of the surfaces themselves.

The mapping of user given parameters or, more informally, *easy access parameters* onto equation model parameters is done by one or more subroutines. The reference to these routines is declared under the heading Parameter Processing.

3.2.6 Flexible Model Descriptions – revisited

Behavioral Flexibility. Increased behavioral flexibility is provided in the new environments by the possibility of locally replacing component or subsystem models without disturbing the model as a whole. This feature is most frequently used to experimentally find an appropriate level of approximation for a particular submodel. Behavioral flexibility is often needed at run time as well. The internal description of a model may need to change significantly as certain conditions are fulfilled. A typical example is when a component goes into a saturated state, e.g., when a heating coil or a fan has reached its capacity limit, or when a model has a singularity, which can be circumvented by a local re–description.

Changes like these can of course be hidden in external functions, like we did with the ventilation air demand equation of the mh_zone model. It would however make the model more legible if they were declared explicitly. The construct provided for this in the NMF is the *conditional expression* with a structure familiar to every programmer, for example:

```
/* required supply air mass flow rate in the mh_simple_zone */
m = IF temp >= tmax THEN
      mmin
ELSE IF temp <= tmin THEN
      mmax
ELSE  (mmin – mmax)*temp/(tmax – tmin)
END IF
```

3.3 Algorithmic Models

Although continuous elements form the bulk of a building simulation model, algorithmic models operating in discrete time are more suitable for certain components. Sampling, micro processor based controllers are obvious physical examples of such components. A further activity in a simulation that lends itself to algorithmic description is boundary data processing. To calculate, for example, the amount of solar radiation that falls on a certain surface at a particular time of day is more straightforward in algorithmic form. The inherent input–output orientation of an algorithmic description is in these cases only a minor restriction, since one rarely is interested in the reverse question: What input gives rise to this output?

For these reasons, algorithmic components are next on the list of items to be formatted or standardized. Due to space constraints we omit any further discussion about this here.

4. SUPPORTING SOFTWARE

Part of the NMF concept is also a set of supporting software tools. Among these are translation tools for conversion of the models from the standard format to that of particular modeling environments. Also needed are tools for the creation and maintenance of component models, and those for the creation and maintenance of libraries of such models. It is envisioned that some organization will host a "base" library. This base library and the software tools can be ported to any modeling environment. It is anticipated that once ported, the base library will be augmented as required by the needs of the user.

The library itself contains models in the standard format. The translation software should be partitioned into the portion that will be the same regardless of the target environment, called the interpreter, and a portion that generates the models in the format of a particular environment, called the generator. The interpreter knows the standard syntax and library format, while the generator knows the syntax and format of the target environment. Obviously, the interpreter can be delivered with the library, while the generator will have to be customized for each environment.

This is not unlike the task of porting systems software to different hardware environments. It can be made relatively straightforward by providing implementers with example generators for well–known environments, and certain software tools that are commonly needed.

5. CONCLUSIONS AND DISCUSSION

The NMF has been proposed as an alternative to crafting component models for each of the existing and evolving simulation environments. The NMF has the following precepts: (1) Essential information about the component models is formalized in order to allow automatic translation; (2) Continuous models are equation based; (3) The concept of typed links encourages libraries of plug–compatible component models; (4) Mathematical validation is required for each model. We have shown some of the details of the NMF specification in a few examples. However, it must be understood that as further component models are developed in the format, changes and additions of this preliminary proposal may be required.

A key part of the concept is automatic translation. Feasibility of translation to a specific environment has been demonstrated for SPANK using MACSYMA [BUHL 1989]. Furthermore, generation of Ida models is straightforward, since they are also equation based. Type routines for TRNSYS and HVACSIM+ involve an additional difficulty because algebraic equations are solved locally within each subroutine. A general purpose non–linear algebraic equation solver will have to be part of the generators for these environments. There are however several robust solvers readily available in e.g. the SLATEC library.

Given a good generator, the format should provide a TRNSYS component modeler with a time effective alternative to direct Fortran programming.

Perhaps the most significant potential outcome of the NMF would be the inception and growth of a significant public domain library of building component models. For this to occur, six events are required:

1. The acceptance of a standard format, perhaps based on the one outlined herein.

2. The development of the software that interprets the standard format.

3. The development of a model generator for each of several widely used environments, e.g., TRNSYS, and HVACSIM+.

4. The development of software tools for the creation and maintenance of a library.

5. The creation and validation of an initial base library containing frequently used component models, such as those used for energy analysis. The TRNSYS library could be a starting point.

6. Establishment of a mechanism for acceptance of new models for the base library, as well as formation of new specialized libraries.

Once these events have taken place, there should be be a strong incentive to use and extend the library, both because of the economy relative to independent library development by the user communities of each environment, and because of the desire of users to employ accepted component models. Once the library comes into wide use, forces will develop to extend it into new areas, such as control simulation.

ACKNOWLEDGEMENTS

The basic ideas behind the NMF came forth during a series of discussions at the Swedish Institute of Applied Mathematics. The authors have merely put it all on paper. Magnus Lindgren, Axel Bring, Lars Eriksson and Gustaf Söderlind have been participating in these discussions in addition to the authors.

REFERENCES

BUHL 1989 W.F. Buhl, E.F. Sowell, J.M. Nataf "Object Oriented Programming, Equation Based Submodels, and System Reduction in SPANK," Building Simulation '89 Conference, Vancouver

CLARKE 1984 J.A. Clarke, L. Laret "Explanation of the Data Processor Proforma," ABACUS, Strathclyde, and Laboratoire de Physique du Batiment, Liege, working document, Dec., 1984

CLARKE 1985 J.A. Clarke, J.J. Hirsch, W.F. Buhl, A.E.Erdem, F.C.Winkelmann, E.F.Sowell, A. Lahellec, N. Huang, J. Sornay, L. Laret "A Proposal to Develop a Kernel System for the Next Generation of Building Energy Simulation Software." Lawrence Berkeley Laboratory, Nov. 1985

CLARKE 1986 J.A. Clarke "The Energy Kernel System: A Technical Overview," Proceedings of the Second International Conference on System Simulation in Buildings. Liege, Dec., 1986

DUBOIS 1988 A.M. Dubois "MODEL–BASED COMPUTER AIDED MODELLING: the new perspectives for building energy simulation," communication from CSTB, B.P. 21, 06561 VALBONNE Cedex, France

ELMQVIST 1986 H. Elmqvist "LICS: Language for Implementation of Control Systems," Dept. of Automatic Control, Lund Institute of technology, Box 118, 221 00 Lund, Sweden

MATTSSON 1988 S.E. Mattsson "On Model Structuring Concepts," Presented at 4th IFAC Symposium on Computer Aided Design in Control Systems (CADCS), Beijing, China

PETZOLD 1982 L.R. Petzold "A Description of DASSL: A Differential/Algebraic System Solver," Proceedings of IMACS World Congress, Montreal, Canada, 1982

SAHLIN 1988 P. Sahlin. "MODSIM: A Program for Dynamical Modelling and Simulation of Continuous Systems." Proceedings of the 30th annual meeting of the Scandinavian Simulation Society, ISSN 0357–9387

SOWELL 1986 E.F. Sowell, W.F.Buhl, A. E. Erdem, and F.C. Winkelmann. "A Prototype Object–based System for HVAC Simulation." Proceedings of the Second International Conference on System Simulation in Buildings. Liege, Dec., 1986

SÖDERLIND 1988 G. Söderlind, L.O. Eriksson, A. Bring "Numerical Methods for the Simulation of Modular Dynamical Systems," Report from the Swedish Institute of Applied Mathematics