FUTURE TRENDS OF THE NEUTRAL MODEL FORMAT (NMF)

Per Sahlin[‡], Axel Bring[‡], and Kjell Kolsaker[†]

KTH, Stockholm, Sweden, and NTH, Trondheim, Norway

ABSTRACT

The Neutral Model Format for building simulation was proposed in 1989 as a means for documentation and exchange of models. It has attracted much interest and an acceptance as a potential standard, maintained by ASHRAE's TC 4.7 technical committee. So far, the format has only been directed towards component (leaf) models, but many suggestions have been made to extend it to also cater for systems of component models. A brief review of NNE is given. This paper makes detailed proposals for NNE extensions covering hierarchical modeling, inheritance, equation-based function definitions, link types with associated equations, and a general method to handle non-standard extensions. A future development of NNE towards STEP/EXPRESS is also discussed.

1. INTRODUCTION

Most researchers in building simulation agree that the present technology - with a large number of stand-alone monolithic Building Performance Evaluation (BPE) tools - is unlikely to ever satisfy constantly changing industrial demands. An entirely new approach is called for. One solution is provided by the so called object oriented simulation environments (OOSE), several of which are available or under development. These new modular systems offer radically increased levels of adaptability, while customizable user interfaces ensure high quality end-user tools. For developers, the production time of new BPE tools is drastically reduced and the resulting tools can be efficiently maintained. For end-users, the main advantage is realistic access to a range of related BPE tools, that have uniform interaction principles and that are data compatible with each other.

A common characteristic between OOSE's is that physical models are regarded as data; they are not regarded as such in most present tools, but are generally inextricably bound to the program code. In OOSE's, submodels may easily be added, changed, removed and combined arbitrarily. This is where the real power of the new systems lie. However, it also creates a common need for well validated and documented models, preferably expressed in a standardized machine-readable way. In fact, without a comprehensive, validated library of ready made component models in a relevant application area, most simulation environments are of limited usefulness. To develop all necessary models from scratch is, in most projects, quite unrealistic. And since the cost of developing a substantial library easily exceeds the development cost of the simulation tool itself, it is important to be able to reuse what other people have already done. This was the basic motivation for proposing a text based neutral model format to the building simulation community in 1989 [Sahlin and Sowell 1989]. Since then the proposal has attracted a great deal of interest from environment developers and users in several application fields. Translators have been developed for IDA [Kolsaker 1994a], SPARK [Nataf 1994], ESACAP (prototype) [Pelletret 1994a], TRNSYS, HVACSIM+ [Sahlin 1995], and MS1 [Lorenz 1994]. Export and import capabilities are planned for ULM (ALLAN) [Jeandel 1994].

Pending formal standardization, ASHRAE (American Society of Heating, Refrigerating, and Air-conditioning Engineers) has formed an ad hoc committee that approves changes to the present format.

NMF has two main objectives: (1) models can be automatically translated into the local representation of several simulation environments, i.e. the format is program *neutral* and machine readable; and (2) models should be easy to understand and express for non-experts. The first objective enables development of common model libraries, which can be accessed from a number of simulation environments.

Section 2 gives a brief introduction to NMF. The proposed extensions are presented in section 3 and illustrated with numerous examples. Section 4 suggests advantages attainable by a STEP [ISO TC 184 1993] orientation of future NMF development.

2. NMF REVIEW

Internal component model behavior is described by a combination of algebraic and ordinary differential equations. Equations may be written in any order and in the form

<expression> = <expression>;

NMF only *states* equation models, while *solution* of equations is, in some cases, left to the target environment (e.g. IDA, or SPARK), or the NMF translator in others (e.g. TRNSYS, or HVACSIM+).

NMF supports model encapsulation through a link concept, i.e. models may only interact via variables appearing in LINK statements. To enhance and encourage model plug compatibility, links and variables are globally typed. The idea is that a basic list of such types should be included in each revision of the standard, but

‡ Dept. of Building Services Engineering,
Royal Institute of Technology,
100 44 STOCKHOLM, SWEDEN
phone: +46-8-411 32 38, fax: +46-8-411 84 32,
e-mail: abring or plurre@engserv.kth.se

† NTH/ VVS-teknikk
7034 TRONDHEIM, NORWAY
phone: +47-7-59 25 09, fax: +47-7-59 38 59
e-mail: Kjell.Kolsaker@termo.unit.no

that users may add to the list as need arises. A selection of such global types is located in Section 3.6.1.

A quantity type includes a physical unit and information about potential (across) or flow (through) type. A link type is simply an ordered list of quantity types. An example of an NMF model using the heat equation in one dimension can also be found in Section 3.6.1.

To enable direct model translation to input-output oriented environments (e.g. TRNSYS, or HVACSIM+), variable declarations have a role attribute indicating IN for given variables and OUT for calculated ones. Variables and parameters may be scalars, vectors, and matrices. A parameter is anything that must remain constant throughout every simulation. Links may also be vectors, thus allowing models with variable number of ports. Vector and matrix dimensions are governed by a special type of parameter, model parameters. Regular and model parameters are divided into two categories, user supplied and computed, algorithmic computation of which is described in the parameter processing section.

Arbitrary foreign functions in Fortran 77 or C may be defined, either globally or locally within a model. Special functions are defined to handle discontinuities, hysteresis, linearization, and errors. A more complete account of NMF is given in the reference report [Sahlin, Bring, and Sowell 1994[1]]

3. PROPOSED EXTENSIONS

Ever since the original NMF paper there has been a healthy flow of proposals for new features. The bulk of these have concerned additional support for good model and code structure. Some have suggested syntax for extensions that were already implicitly defined in terms of existing constructs; others take up entirely new threads of thought. It seems that we now have enough material in terms of modeling experience and proposal "raw material" to decide on a well balanced set of extensions, well in tune with the original NMF ideas. This section will outline our view of a package of such extensions: hierarchical modeling on the link level; function definitions that may contain implicit equations and that can be symbolically processed; property links - link types with associated equations; model inheritance; and a standardized gateway for solver specific extensions.

3.1 Hierarchical Modeling

Since the most imminent need for model exchange between environments is on the component rather than on the system level, no standardized syntax for system models was included in the original NMF proposal. System models would, for the time being, be handled separately in each target environment. This decision has been criticized by many and there has consequently been a number of proposed system modeling constructs. Developers that are looking for a complete modeling language clearly need a standardized way of expressing system models. It might seem less obvious that a language, mostly intended as a vehicle for construction of component model libraries, would need a system modeling capability. However, the division between what is a component and what is a system is generally so vague that it becomes unreasonable to base any language limitations on it. The arguments in favor of inclusion of system modeling features seem quite convincing. The next issue is to determine a good range of hierarchical modeling constructs.

Most obvious is a system model construct, with references to underlying submodels, which of course might be system models themselves, and the possibility to interconnect submodels at the link rather than the individual variable level. This type of model is clearly implied already in the existing link construct. We will use the standard NMF two component system - a "Heating Collector" (fig. 1) for mixing and heating two air streams - to illustrate the suggested syntax.

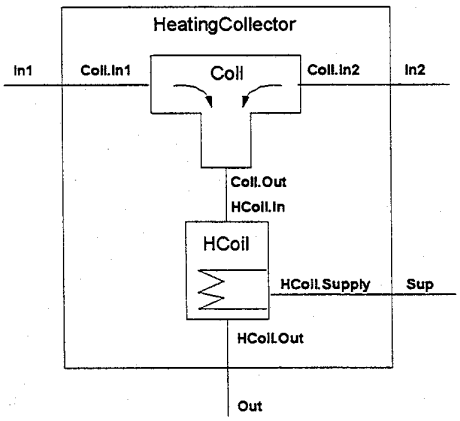

Fig. 1. System model with a collector and heating coil

```
SYSTEM_MODEL Heating_Collector

PARAMETERS
    HeatFlux    my_flux     S_P     100     "a parameter";

SUBMODELS       // Declare submodel instances
//  class       instance
//      data
    Collector   Coll;
    Heat_Coil   Hcoil,
        rise_time := 30;
        // rise_time is a parameter of Heat_Coil

CONNECTIONS     //Link level internal connections
//  inst.link   =   inst.link;
    Coll.Out    =   Hcoil.In;

LINKS           // Lift (and rename) submodel links to
                // become Heating_Collector links
//  name        =   inst.link;
    In1         =   Coll.In1;
    In2         =   Coll.In2;
    Out         =   Hcoil Out;
    Sup         =   Hcoil.Supply;
```

[1] An ASCII version of the reference report and other NMF material is located at: `ftp://mailbase.ac.uk/pub/lists-f-j/ibpsa-nmf`. Connect as user `anonymous` and give your e-mail address as password.

```
DOCUMENTATION
END_DOCUMENTATION

END_MODEL
```

Submodels are declared (instantiated) in the SUBMODELS Section; we will in this text use the word instantiated for a declared submodel, although the submodel is not physically instantiated until the surrounding system model is. Parameter values and variable initial values may be supplied in order to override defaults in the instantiated model. Internal connections between submodels are made in the CONNECTIONS Section, where variables of joined models are connected collectively at the link level, with appropriate signs for THRU variables.

The LINKS Section defines those submodel links that should be visible on the boundary of the defined system, and thus be available for connection at the next level up in the hierarchy.

A PARAMETER section may be declared for systems. Parameter expressions may be used in the SUBMODELS section.

The present NMF continuous_model maps in a natural way to TRNSYS or HVACSIM+ type routines. The system_model construct that we are suggesting here would enable NMF expression of TRNSYS decks or HVACSIM+ work files as well.

3.2 Hybrid System/Continuous Models

In this extension proposal, equations are not allowed in system_ models, nor are submodel references allowed in present continuous_models. It is in our opinion vital to have a pure system model that only contains submodel references and never deals with individual variables and internal submodel behavior. This allows simplified modeling software, without equation manipulation capability, for interconnection of existing submodels. Examples of such tools are the present front-end programs for TRNSYS and HVACSIM+. It also makes possible top-down modeling, where the precise structure of submodels and connected links remain undecided. So far, NMF only handles equation based submodels, but the intention is to extend this to other model categories, perhaps most imminently discrete time models for sampling controllers etc. The (pure) system_ model, as proposed here, will be able to accommodate submodels of several types.

For simplicity, our initial extension idea was to avoid models that mix submodel references with individual equations - hybrid models - in NMF. Hybrid models are difficult to handle for a TRNSYS/HVACSIM+ translator. Since they may contain any type of equation, they have to be mapped to type subroutines rather than to decks/work files. This means that the hierarchical structure of the system model part of the hybrid model will have to be flattened out. Since the hierarchy might be arbitrarily deep, generated variable and parameter names in the flattened model will be either very long, if you want to keep original names, or else quite cryptic.

In our view, the modeling flexibility gained by allowing hybrid models, does not by itself motivate the necessary increase in translator complexity. However, the required capability to flatten (or expand) hierarchical system models can also be useful in other ways. Most importantly, it would enable a user to optionally expand a system model into a single, flat, continuous model, which in turn could be translated directly into a type subroutine. It would also facilitate translation of NMF system models to differential-algebraic equation (DAE) solvers without any submodel support, such as the popular DASSL solver [Brenan 1989]. Finally, it would simplify general symbolic processing of system models, to e.g. eliminate uninteresting variables, or application of index (of nilpotency) reduction algorithms. For these reasons, we have decided to propose a hybrid model type, a continuous_system_model, that mixes equations with references to submodels, which have to be equation based. The heating collector with a slight variation will illustrate the proposed syntax.

```
CONTINUOUS_SYSTEM_MODEL Heating_Collector_w_eqn

VARIABLES
   HeatFlux        Heat_sup   OUT        "supply power" ;

PARAMETERS
   HeatFlux        my_fluxS_P    100 "a pram" ;

SUBMODELS          //Declare submodel instances
//  class           instance
      // data
   Collector       Coll;
   Heat_Coil  Hcoil,
      rise_time := 30;
      // rise_time is a parameter of Heat_Coil

EQUATIONS
   // Regular equations and NMF assignments
   0 = - Heat_sup + 10*Sin(Time*2*PI/(24*t_scale));
   // may be mixed with link level internal connections
   //  inst.link     =   inst.link;
   Coll.Out        =   Hcoil.In; // (as for pure system models)
   // and arbitrary equations containing submodel
   // variables with syntax (inst . link . var)
   Hcoil.Supply.Q_sup/11.4 = Heat_sup/11.4;

LINKS              // Lifted submodel links or
                   // locally defined links may be mixed
   In_1    =   Coll.In1; // a lifted link, as for pure systems
   MTG             In_2    POS_IN Coll.In2.M2, Coll.In2.T2,
                           Coll.In2.W2;
                   // the parallel link, not lifted but
                   // redeclared. Locally defined variables
                   // (such as Heat_sup) may also appear here
   Out =  Hcoil.Out;       //also lifted

DOCUMENTATION
END_DOCUMENTATION

END_MODEL
```

This hybrid model concept is very similar to that proposed by [Sowell 1994]. Similarly to Sowell's proposal, submodels may only interact with each other via variables appearing on the links of the joined models.

The fundamental difference between our proposal and Sowell's original concept is that it enables link level op-

erations as well as variable level operations, i.e. submodel links may be connected or lifted on the link level rather than always on the variable level. In the example, all the variables of the collector Out link are joined to the heater In variables in one operation. Similarly the collector In1 link is lifted and renamed In_1 in one operation. To illustrate the variable level alternative, the In_2 link of the collector has been decomposed into its individual variables, and then (trivially) assembled again. The latter construction would of course allow a redefinition of the link, by for example adding an extra variable.

Connecting and using vector valued links requires a more detailed specification, which is left out here.

3.3 New Function Definitions

In the present syntax (version 3), substructuring is achieved mainly through the definition of functions. The main purpose of external function definitions was originally to provide a gateway to foreign code, and all functions and subroutines had to be coded in a foreign language. In version 3, NMF assignment modeling is allowed in function definitions as an alternative to C and Fortran and this is appreciated by most modellers, who find it convenient to write everything in a single language. Another advantage is that these NMF based function definitions are as accessible as equations for symbolic processing. The limited statement repertoire of NMF could easily be processed both to generate derivatives and to generate alternative inverses of functions. Since function references are very natural in an equation based context, we see this as a superior alternative in many of the examples that have been presented as motivation for hybrid modeling. Compare for example the collector model with a call to a function in the equations (not through help variables) to the suggested syntax with local links.

Our preferred syntax for equation and link sections, referring to a *function* EnthalpyF(W,T) (not a continuous model):

```
CONTINUOUS_MODEL  CollMoistAir

// documentation and declarations omitted

LINKS
//  type        name        variables
    MTG         In1_air     POS_IN M1, T1, W1;
    MTG         In2_air     POS_IN M2, T2, W2;
    MTG         Ivg_air     POS_OUT Mo, To, Wo;

EQUATIONS
    Mo = M1+ M2;
    Mo * EnthalpyF(Wo,To)
        = M1 * EnthalpyF(W1,T1)
        + M2 * EnthalpyF(W2, T2);
    Mo * Wo = M1*W2 + M2*W2;
```

The corresponding sections in our proposed hybrid syntax with reference to a continuous_model EnthalpyM:

```
CONTINUOUS_SYSTEM_MODEL  CollMoistAir

// documentation and declarations omitted

LINKS
//  type        name        variables
    MTG         In1_air     POS_IN M1, Ent1.Int.T ,
                            Ent1.Int.W;
    MTG         In2_air     POS_IN M2, Ent2.Int.T ,
                            Ent2.Int.W;
    MTG         Lvg_air     POS_IN Mo, Ento.Int.T ,
                            Ento.Int.W;

SUBMODELS
    EnthalpyM       Ent1;   // Instantiation of
                            // CONTINUOUS_MODEL EnthalpyM,
    EnthalpyM       Ent2;   // with a single LINK:
                            // HTG Int H, T, W;
    EnthalpyM       Ento;

EQUATIONS
    Mo = M1 + M2 ;
    Mo * Ento.Int.H =   M1 * Ent1.Int.H
                      + M2 * Ent2.Int.H;
    Mo * Ento.Int.W   = M1 * Ent1.Int.W
                      + M2 * Ent2.Int.W;
```

It seems obvious that the hybrid model alternative is considerably more awkward than the first version, where Enthalpy is treated as a function rather than as a submodel. The fundamental difference between a submodel and a function call is that the latter does not have an internal state, i.e. no internal parameters or state variables are associated with each function call. The restriction to only algebraic equations eliminates state variables. Thus, we can only speak of instantiation with respect to submodels and never to functions.

Kolsaker [Kolsaker 1994c] has suggested improved declaration of INPUT and OUTPUT quantities of functions, making them look more like the corresponding declarations in continuous models and to enable unit checking. This seems to be a very sound alternative, although not backwards compatible. Perhaps the old version, which has its merits in terms of simplicity, could coexist.

Going one step further regarding function definitions, we propose to allow algebraic equations among function assignments.

```
VOID FUNCTION AirPsych (P, M, T, W, PSat, H)

DOCUMENTATION
Definition of psychrometric relations for moist air
END_DOCUMENTATION

QUANTITIES
//  type          name    role    description
    Pressure      P       IN      "air pressure" ;
    MassFlow      M       IN      "not used in eq" ;
    Temp          T       IN      "dry bulb temp" ;
    HumRatio      W       IN      "humidity ratio" ;
    Pressure      PSat    OUT     "saturation pressure";
    Enthalpy      H       OUT     "enthalpy" ;

CODE
    // equations and assignments may both occur.
    // Call function to calculate saturation pressure
    PSat := ASHRAE_SaturationP (T) ;
    0 = -H + CP_Air * T + W * (CP_VAP * T + HF_Vap) ;

END_CODE
```

```
END_FUNCTION
```

Quantities acting both as inputs and outputs (A_S variables in the calling model), receive role I_O. Local quantities are declared LOC. Variables of any role may appear in equations, and all except IN may be assigned to once, i.e. several assignments of the same variable may occur, if in the same conditional statement and excluding each other. Functions that return a value are defined similarly, but with an output quantity type instead of VOID. Such functions may not in general have I_O variables.

These rules are not in complete harmony with current role definitions in continuous models. Space will not allow a thorough digression on these matters here. However, it is possible (and preferable) to change, in a backwards compatible fashion, the current role definitions in continuous models to be more adequate and at the same time to be in harmony with the definitions above.

The new features we have so far discussed, the system models and the extended function declaration, form a natural set of changes that should satisfy the most imminent needs for hierarchical modeling. However, we would like to go further and address two additional structural problems.

3.4 Property Links

NMF link types are ordered sets of typed variables and normally correspond to physical connections having well defined properties. Examples are fluid flows, electrical connections, control signals (physical or logical). When defining link types, it is desirable to select 'minimal' sets of variables, sufficient to define the media properties relevant for the modeling level of the connection. Dry air flows could e.g. be described, either by (massflow, pressure, temperature) or (massflow, pressure, enthalpy) but preferably not by the redundant set (massflow, pressure, temperature, enthalpy). Component models using these link types will, however, often make use of quantities that have to be derived from the minimal set in the link. An example is a collector with temperatures in the links but expressing enthalpy conservation.

Much of the discussion on NMF extensions has centered on how these (missing) property equations can be handled in component models. We propose that *link property models* be introduced to handle this.

Extended link types will then contain a (minimal) set of variable types, plus a link model defining further variables and their relations (to the minimal set). Component models can optionally use the extra variables by referring to the link model, but without repeating the link model equations. Duplication of equations is thus avoided in the description of component models with common property links. NMF translators will automatically add property equations and variables as necessary in generated component models.

We think that the basic information about such extended link types should be presented in a very compact way among the present global link type definitions, in order to retain the index character of this section. The actual models, which in our opinion should be limited to assignments and algebraic equations, could very well be declared in terms of a new-style subroutine (VOID FUNCTION) definition. The basic declaration among the global link types looks like this:

```
FatMoistAir (   Pressure "air pressure",
                MassFlow,
                Temp,
                HumRatio )
    DERIVED_FROM air_psych
            (   Pressure "saturation pressure",
                Enthalpy ) ;
```

The subroutine air_psych would take the four first variables as input and calculate the remaining two as output. The order of the variables in the formal argument list of air_psych would be the same as in the link type declaration.

The equation and link sections of a moist air collector would then be:

```
EQUATIONS
    Mo = M1 + M2 ;
    Mo * Ho = M1 * H1 + M2 * H2 ;
    Mo * Wo = M1 * W1 + M2 * W2 ;

LINKS
//  type            name        variables
    FatMoistAir     In1_air     P, POS_IN  M1, T1, W1,
                                VOID, H1;
    FatMoistAir     In2_air     P, POS_IN  M2, T2, W2,
                                VOID, H2;
    FatMoistAir     Lvg_air P, POS_OUT Mo, To,
                                Wo, VOID, Ho;
```

3.5 Model Inheritance

Another feature that is missing in NMF is a possibility to exploit similarities between related models. An obvious way to handle this is via inheritance, in the manner of OOP. The most important reasons for an NMF class structure are that we may express information about a compatible family of models in a single place, and that trivial specializations of a model may be expressed separately, e.g. different parameter processing for different component manufacturers etc.

The single inheritance scheme we are suggesting obeys the following rules:

I. New declarations under an NMF heading are added after previous (inherited) declarations

II. Uniquely identifiable declarations may be overridden, except for assignments, which are never overridden.

This means that, for the case of continuous models, equations and assignments may be added to previously defined ones. Overriding previously made assignments is in some cases technically possible, but would result in cryptic code. Redeclaring a previously declared link, variable, or parameter, cancels the previous declaration

completely. Overriding will be used mostly for link declarations, to provide increasingly detailed link types.

A new keyword CLASS is introduced. A CLASS (in our definition) may *not* be instantiated. It is used only for structuring purposes, as a repository for class-common definitions.

Multiple inheritance is excluded. Its merits in this context are questionable, and it may result in declaration collisions. These could be resolved by allowing e.g. aliasing of inherited properties or by letting the order of the inheritance list carry meaning. However, since we are striving for simple solutions, our suggestion is that single inheritance is sufficient for the moment.

As in illustration of the technique, we can choose collector models.

```
CLASS CollTemplate

DOCUMENTATION
END_DOCUMENTATION

LINKS
//   type        name        variables
     MT          InAir1      POS_IN M1, T1 ;
     MT          InAir2      POS_IN M2, T2 ;
     MT          OutAir      POS_OUT Mo, To ;

EQUATIONS
     // mass balance
     Mo = M1 + M2 ;

VARIABLES
     MassFlow    M1    IN     "mass flow 1 in"
     MassFlow    M2    IN     "mass flow 2 in"
     MassFlow    Mo    OUT    "mass flow out"
     Temp        T1    IN     "temp of flow 1"
     Temp        T2    IN     "temp of flow 2"
     Temp        To    OUT    "temp of flow out"

END_MODEL
```

The template is not a complete model by itself; at least enthalpy balance has to be added.

```
CONTINUOUS_MODEL (CollTemplate) DryColl

EQUATIONS
     // add enthalpy balance
     Mo * CP_AIR * To = M1 * CP_AIR * T1
                      + M2 * CP_AIR * T2 ;

END_MODEL
```

The template can also be used to define a collector for moist air.

```
CONTINUOUS_MODEL (CollTemplate) MoistColl

LINKS                  // redefine all three links
//   type        name        variables
     MTG         InAir1      POS_IN M1, T1, W1 ;
     MTG         InAir2      POS_IN M2, T2, W2 ;
     MTG         OutAir      POS_OUT Mo, To, Wo ;

EQUATIONS
     // add enthalpy and humidity balances
     Mo * EnthalpyF(To,Wo)
          = M1 * EnthalpyF(T1,W1)
          + M2 * EnthalpyF(T2,W2) ;
     Mo * Wo = M1 * W1 + M2 * W2 ;

VARIABLES
     // add declarations for humidities
     HumRatio    W1 IN       "hum ratio of flow 1"
     HumRatio    W2 IN       "hum ratio of flow 2"
     HumRatio    Wo OUT      "hum ratio of flow out"

END_MODEL
```

Now, let us try to decompose the Heating_Collector using inheritance.

```
CLASS HC_template

DOCUMENTATION
Common doc's go here
END_DOCUMENTATION

SUBMODELS              // this section is (optionally)
                       // declared here for clarity only
//   GENERIC      instance
     GENERIC      Coll;
     GENERIC      Hcoil;      // no data allowed for
                              // GENERIC

CONNECTIONS
//   inst.link    =  inst.link ;
     Coll.Out     =  Hcoil.In ;

LINKS
//   name         =  inst.link ;
     In1          =  Coll.In1 ;
     In2          =  Coll.In2 ;
     Out          =  Hcoil.Out ;

END_MODEL
```

One possible use of this resource is of course a new edition of the Heating_Collector:

```
SYSTEM_MODEL (HC_template) Heating_Collector

DOCUMENTATION
Suplementary, class specific, doc's are added here
END_DOCUMENTATION

SUBMODELS
//   class                instance
//     data
     Collector            Coll ;
     Heating_Coil         Hcoil
          Q_max := 1000,      rise_time := 30;

END_MODEL
```

A nice feature is that we easily may populate the HC_template with new submodels, which may have entirely different link structures.

```
SYSTEM_MODEL (HC_template)
                    Moist_Heating_Collector

DOCUMENTATION
END_DOCUMENTATION

SUBMODELS
//   class                          instance
//     data
     Moist_Collector          Coll ;
     Moist_Heating_Coil       Hcoil
```

```
    Q_max := 1000,      Other_data := 4017;
END_MODEL
```

3.6 NMF Extension keyword

Each implementor of NMF based tools has added extensions as necessary for the targeted environments. This is completely in accordance with the original ideas, provided that the NMF part of a full model description remains meaningful to others, and that extensions can be easily removed, leaving standard NMF only. It has become obvious that it would be an impossible endeavor to strictly define what is truly neutral about a model. The only feasible solution is to construct a working process for the successive inclusion of features of sufficient common interest. In principle, the ASHRAE NMF committee provides a forum for additions to the language but individual users and groups of users need access to quicker, more informal, means.

Kolsaker [Kolsaker 1994a] has to this effect suggested a special pair of keywords EXTENSION and END_EXTENSION on the continuous model level, to delimit whatever information, in a free format, that might be useful in a particular context. The only requirement on the delimited information is that the second token should identify the purpose of the following text and be readable by any NMF translator. Aside from being a good pragmatic solution for individual users, the suggested mechanism would work as a source of inspiration for formal extension proposals - a gateway for organic growth.

We propose that extensions may occur in a number of well defined places in the NMF code: among global declarations, in models, and tagged on to individual statements. Any front-end NMF translator should then be prepared to store the unprocessed extension code for subsequent parsing in the appropriate back-ends. Since extensions may appear frequently in target code for a particular solver, we suggest an abbreviated keyword pair: EXT and END_EXT.

3.6.1 NMF Samples with possible extension locations

Extension keyword pairs are marked with ⋙→; only places of principal importance have been selected.

```
QUANTITY_TYPES
//  Quantity types are used for both variables and
//  parameters. For parameters the kind is irrelevant.
//  CROSS = Potential, non-directional
//  THRU = Flow, directional
//  type name          unit          kind */
    Enthalpy           "J/kg"        CROSS ⋙→;
    Factor             "dimless"     CROSS ;
    HeatCapM           "J/(kg K)"    CROSS ;
    HeatCondA          "W/(m2 K)"    THRU ;
    HeatFlux           "W"           THRU ;
    HumRatio           "kg/kg"       CROSS ;

⋙→ //Extensions may occur once at the global level

LINK_TYPES
//typename           (variable types)
TQ                   (Temp, HeatFlux) ⋙→;
MT                   (MassFlow, Temp) ⋙→;
PMT                  (Pressure, MassFlow, Temp) ;
MoistAir             (Pressure, MassFlow, Temp, HumRatio) ;
```

A sample thermal model of a wall:

```
CONTINUOUS_MODEL tq_hom_wall

DOCUMENTATION
A 1-D finite difference wall model. One homogeneous layer. TQ
interfaces on both sides.
END_DOCUMENTATION

⋙→ //Once at the model level

EQUATIONS
    //NMF assignments can always be interpreted as if
    //they were equations
    Taa := 2 * Ta - T[1] ⋙→;
    Tbb := 2 * Tb - T[n] ;

    // space discretized heat equation, T' indicates dT/dTime
    FOR i = 2, (n-1)
        c_coeff * T'[i] = T[i - 1] - 2. * T[i] + T[i + 1] ⋙→;
    END_FOR ;
    c_coeff * T'[1] = Taa - 2.* T[1] + T[2] ⋙→;
    c_coeff * T'[n] = T[n - 1] - 2. * T[n] + Tbb ;
    // boundary equations
    0 = -Qa + d_coeff * (Taa - T[1]) ;
    0 = -Qb + d_coeff * (Tbb - T[n]) ;

LINKS
//  type      name      variables
    TQ        a_side    Ta, POS_IN Qa ⋙→;
    TQ        b_side    Tb, POS_IN Qb ;

VARIABLES
//  type      name   role   description
    Temp      T[n]   OUT    "temperature profile" ⋙→ ;
    Temp      Ta     OUT    "a-side surface temp" ⋙→ ;
    Temp      Tb     OUT    "b-side surface temp" ;
    Temp      Taa    LOC    "a-side virtual temp" ;
    Temp      Tbb    LOC    "b-side virtual temp" ;
    HeatFlux  Qa     IN     "a-side entering heat" ;
    HeatFlux  Qb     IN     "b-side entering heat" ;

MODEL_PARAMETERS
//  type  name  role  [defmin max]  description
    INT   n     SMP   5   3   BIGINT "number of temp
layers" ⋙→ ;

PARAMETERS
//  type          name      role   description
//  supplied parameters
    Area          a         S_P    "wall area" ⋙→ ;
    Length        thick     S_P    "wall total thickness" ;
    HeatCondL     lambda    S_P    "heat transfer coeff" ;
    Density       rho       S_P    "wall density" ;
    HeatCapM      cp        S_P    "wall heat capacity" ;
//  computed parameters
    generic       d_coeff   C_P    "lambda*a/dx" ;
    Length        dx        C_P    "layer thickness" ;
    generic       c_coeff   C_P    "rho*cp*dx*dx/
                                    (lambda*t_scale)" ;

PARAMETER_PROCESSING
    dx := thick / n ⋙→ ;
```

```
    c_coeff := rho * cp * dx * dx / (lambda * t_scale) ;
    d_coeff := lambda * a / dx ;
END_MODEL
```

4. STEP RECONCILIATION

Since the first proposal in 1989, the discussion about various NMF-constructs has focused on grammar, as indeed this paper does as well. The textual appearance of selected models has been the main object. This is of course quite appropriate for the equation core of component models, but for instantiated system models and related data it may be fruitful to also regard data models directly, and to treat the textual representation as one of several possible views. STEP/EXPRESS [ISO TC 184 1993] seems to be an appropriate vehicle for an extension of the future NMF discussion in this direction.

Further reasons for the employment of STEP technology for NMF include:

- STEP dominates product model applications, which will be an important source for generation of simulation models, and these should therefore be encompassed by STEP, either as pure aspect models or as parts of global models
- Many existing STEP/EXPRESS resources will be useful for development of NMF-oriented application tools

In our initial work in this area [Sahlin and Johansson 1994], we have chosen to focus only on the problem of defining conceptual models of NMF instances, and of hierarchical systems of such instances, i.e. data models corresponding to the system_model construct proposed in this paper. The textual description of the data, found in a STEP physical file, would simply be another completely equivalent representation, less appealing to the human eye, but processable by standard STEP tools. In the first set of conceptual models, only the state of a model is encompassed, represented by its quantity values, whereas the internal component behavior, represented by the equations, is not treated. It is assumed that underlying NMF continuous_models are known to all parties.

Another interesting issue is of course the models of internal behavior as well, i.e. to model the present NMF in EXPRESS, with entities such as equation, if_then_else_clause, etc. Such models are necessary for development of NMF parsers and translators.

The main motivation for remodeling the present NMF in EXPRESS is completeness. New component models could be communicated with the same tools and protocols. A potential EXPRESS-based STEP NMF standard would not have to rely on an additional non-EXPRESS standard. Additional benefits can be expected for design and implementation of NMF component model databases and management tools. The present conclusion is that it would be worthwhile to model the present NMF in EXPRESS.

5. CONCLUSIONS

NMF seems to be well on the way of becoming a commonly accepted format for exchange of simulation models. The ASHRAE committee presently in command provides, for the time being, an adequate neutral forum for the development process. We believe that the existence of a standard will be a salient feature in the evolution of object oriented simulation environments, which in turn will provide the building sector with appropriate and powerful simulation tools.

REFERENCES

Brenan, K.E., S.L. Campbell, and L.R. Petzold, 1989 "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations", North Holland

ISO TC 184 1993. "The STEP Standard", draft international standard DIS 10303, continuously since 1992 published in several different parts

Jeandel A. 1994, Personal communication

Kolsaker, K. 1994a. "NEUTRAN-supported NMF Enhancements", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Kolsaker, K. 1994b. "Simpler NMF Description of Advanced Models Using Hierarchical Modelling and Data Abstraction", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

Kolsaker, K. 1994c "NEUTRAN - A Translator of Models from NMF into IDA and SPARK", Proceeding to the BEPAC conference, BEP'94, York

Lorenz F. 1990. "Brief Description of the MS1 (Modelling System 1) Project", private communication

Lorenz F., 1994, "Comments on the Neutral model Format", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Nataf J.-M. 1994, "Translator from Neutral Model Format to SPARK", draft paper presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R. 1994, Personal communication

Sahlin, P, E.F. Sowell 1989, "A Neutral Format for Building Simulation Models", proc. of Building Simulation '89 , Vancouver, BC, International Building Performance Simulation Association

Sahlin, P., A.Bring, E.F.Sowell 1994, "The Neutral Model Format for Building Simulation", Version 3.01, ITM report 1994:4.

Sahlin, P., C. Johansson, 1994 "NMF-Based Aspect Models in STEP for Building and Process Plant Simulation", proc. of the CIC W78 workshop on computer integrated construction, Aug. 22-24, 1994, VTT, Helsinki, Finland

Sahlin, P. 1995. Dec 94 - Feb. 95 "Progress Report on TRP-839 Development of a Component Model Translator for the Neutral Model Format (NMF)", *Report submitted to ASHRAE*

Sowell, E.F. 1994. "A Proposal for Hierarchical Submodels in NMF", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994