NMF HANDBOOK

An Introduction to the Neutral Model Format

NMF version 3.02

Nov 1996

Per Sahlin
Building Sciences
KTH
100 44 STOCKHOLM
plurre@engserv.kth.se

ASHRAE RP-839

Contents

NMF Model Examples

Figures

1. About This Text

The NMF Handbook provides guidelines for modelling with the Neutral Model Format. It is intended to be a first text on the subject, but some topics should also be useful for an NMF modeller with previous experience. The reader is assumed to have mathematical modelling experience and also to be familiar with the general software and hardware tools that are necessary for NMF modelling, such as a compiler and a target simulation environment. NMF is normally used in conjunction with a specific simulation environment, such as TRNSYS. However, this text is environment independent, except for some exercises, where an IDA Solver file is used as a concrete example. The text is a complement to the NMF report *The Neutral Model Format for Building Simulation* and many grammatical details of NMF and other specific rules are not repeated here. The following material is useful to have at hand during reading:

1. The ASHRAE NMF Translator program and associated sample NMF files. This material is available at ftp://urd.ce.kth.se/pub/rp839/.
2. The NMF Report *The Neutral Model Format for Building Simulation*, available at ftp://urd.ce.kth.se/pub/reports/nmfre302.ps.
3. An NMF compatible simulation environment and an associated NMF translator for model testing, available, for example, at http://www.brisdata.se/.

This document is also available in electronic form in Rich Text Format ftp://urd.ce.kth.se/pub/reports/handbook.rtf and in Postscript ftp://urd.ce.kth.se/pub/reports/handbook.ps.

2. Introduction

The author of this handbook has for some time worked with new simulation techniques and languages for continuous modular systems. These techniques are applicable to a large class of static and dynamical simulation problems in, e.g., the building, energy and process industries. One important aspect of this work has been involvement in the definition of the Neutral Model Format (NMF), for expression of component level simulation models.

In the present version of NMF, *component models* (primitive models) are automatically translated from NMF to the proprietary format of the target simulation environment. For example, an NMF model of an axial fan is used to generate an axial fan class in, e.g. IDA, or a corresponding type subroutine in TRNSYS or HVACSIM+. The class is then instantiated in the target environment. The instances are furnished with suitable parameters, and incorporated into a system model. System modelling is not encompassed by the present version of NMF, i.e. TRNSYS *decks* cannot be generated automatically at this stage, only *types*.

In the next two sections we will give a brief overview of current work on modular simulation methods, mainly in the context of building simulation, and of the background of the Neutral Model Format. These sections may be omitted without loss of continuity.

2.1 Modular Simulation Environments

Physical systems that are simulated in Modular Simulation Environments (MSEs) are modular in nature, i.e. they naturally decompose into subsystems. Frequently, identical subsystems are repeated a number of times in a model, a fact that is taken advantage of in many tools. Furthermore, the systems should have a basically continuous behavior, meaning that equations used to describe them, as well as forcing functions, will have a limited number of discontinuities. Purely event driven systems are excluded.

If characterized by equations, the physical systems under consideration will require both algebraic and differential equations. Differential equations can be either ordinary (ODE) or partial (PDE), although current tools, and the present NMF, require that PDEs are explicitly discretized in space and thus turned into ODEs. Note that in contrast to many widely used commercial tools, the simulation environments we are concerned with here are not limited to ODEs only. They allow a free mixture of algebraic and ordinary differential equations generally referred to as differential-algebraic systems of equations (DAE).

Furthermore, the simulation tools under discussion are rarely used for applications where a strict formalism for generating governing equations exists. In, e.g., electrical circuit analysis, multibody mechanics, or structural analysis special purpose systems may be more advantageous.

Examples of physical systems that fit this description can be found in many fields. Chemical process plant simulation is a significant area of application. Energy distribution networks and plants is another. The author of this handbook has mainly

worked with building related systems and important applications within this field are: thermal processes in walls and spaces; air and water based distribution systems and plants; and automatic control.

2.1.1 Separation of Modelling and Solving Activities

In contrast to mainstream design tools in, e.g., building simulation, MSEs separate strictly between the modelling and subsequent system solution activities. A modelling tool is often used for model formulation. This tool generates a system model, generally expressed in a *modelling language*. The model is then treated by a solver. An important benefit of a separate solver is that it may be altered or even exchanged with minimal interference with the modelling environment. Some MSEs rely on regular programming languages as part of their system model description. For these, component models are typically described as subroutines with prescribed structure, while interconnection of pre-programmed component models into system models is described with a dedicated language. Other environments have complete modelling languages, which describe component as well as system behavior.

Key characteristics of the modelling language, such as expressiveness and level of standardization, are critical to the usefulness and development potential of the overall MSE. The Neutral Model Format is part of such a modelling language. NMF can be translated into *subsets* of several complete target modelling languages; it does not cover all constructs in any of the present targets, just a sufficient "common denominator."

2.1.2 Target Users and Software Structure

Most of the simulation tools under discussion are intended for quite sophisticated users, who are well versed in mathematical modelling, numerical methods and advanced use of computers. These tools are not directly suited for designers, without special simulation expertise, that use simulation as one of several methods for design evaluation. However, for the expert, they generally provide an efficient environment for model building, simulation and analysis.

Other tools, e.g. EKS and IDA, are primarily intended for efficient design tool production, and the normal end user will rarely interact directly with the underlying MSE techniques.

2.1.3 Available and Emerging MSEs

A few tools and environments with the discussed main characteristics are already matured and available and others are under development:

TRNSYS was developed during the seventies at the Solar Energy Lab at the University of Wisconsin. It was one of the first modular simulation solvers for DAEs and it is distributed as a Public Domain product. It has recently been furnished with a simultaneous solver, i.e. a solver which solves all equations simultaneously. Several compatible commercial modelling tools have been developed, e.g. PRESIM (http://www.engr.wisc.edu/centers/sel/trnsys/index.html).

HVACSIM+ is a solver with similar characteristics as TRNSYS in terms of model format and structure, but more recent numerical techniques than in the original TRNSYS

are utilized. It was developed by NIST in Maryland and released in the mid eighties on a Public Domain basis [Clark 1985]

SANDYS is a general DAE solver and textual modelling environment developed by ASEA, Sweden, in the early eighties. It is commercially available from ABB Corporate Research [Ohlsson 1991].

ALLAN.Simulation is a graphical modeller and solver combination developed by Gaz de France and CISI Engineering. It is since a few years commercially available from the developers [Jeandel 1993].

ESACAP is a DAE solver by Elektronikcentralen in Denmark. It is commercially available from STANSIM, Denmark.

DYMOLA is a commercial modelling tool with symbolic algebra capabilities and interfaces to several solvers. Available from DYNASIM, Lund, Sweden (http://www.dynasim.se/).

CLIM 2000, a graphical modelling tool for building applications, is developed by Electricite de France for internal use [Bonneau 1993].

IDA, a graphical modelling environment (IDA Modeller) and solver (IDA Solver). The latter is available from Bris Data AB, Stockholm, Sweden. IDA Modeller will be released during 1996 [Bring 1995].

MS1 is a graphical multi input language modeller with interfaces to several solvers by Lorenz Simulation, Liege, Belgium in cooperation with Electricite de France [Lorenz 1990].

ISE, a graphical programmable front end that can be used with several building simulation engines such as TRNSYS and COMIS. A TRNSYS application (IIisibat) is available from the developers at CSTB, France [Pelletret 1995].

SPARK is a solver and graphical model editor under development at LBL, Berkeley, California [Buhl 1993].

OMSIM is a graphical modelling tool under development at the Dept. of Automatic Control at the Lund Institute of Technology, Sweden (http://control.lth.se/~cace/).

EKS is a C++ toolkit for development of energy related simulation design tools, by among others the Univ. of Strathclyde, Scotland [Clarke 1993].

SMILE, a general purpose differential-algebraic modelling and simulation environment developed primarily for energy related problems. Modelling language, model library, and related software is under development at the Technical University of Berlin (http://www.cs.tu-berlin.de/~smile/synopsis.html).

THUVAC is a graphical modelling tool and solver for simulation of HVAC related modular systems. It is under development at Dept. of Thermal Energy, Tsinghua Univ., Beijing, China [Yi Jiang 1994]

2.2 The Neutral Model Format

Without a comprehensive, validated library of ready made component models in a relevant application area most simulation environments are rather useless. To develop all necessary models from scratch is, in many 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 already have 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 SPARK [Nataf 1995], IDA [Shapovalov 1995, Kolsaker 1994c], ESACAP [Pelletret 1994a], TRNSYS [Grozman 1996], HVACSIM+ [Grozman 1996], and MS1 [Lorenz 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.

2.3 Current Development

The present version of NMF (3.02) is fully functional, but the list of desirable new features is nevertheless long. Since the original NMF paper in 1989, several different directions of further development have been proposed, some of which are listed below:

Constructs for hierarchical (system) modelling	One of the more obvious missing items in the current NMF is the possibility to express whole system models, and systems that are in turn (hierarchically) built from other systems. The NMF Committee has approved the basic principles of a recent proposal in this area [Sahlin, Bring, and Kolsaker 1995]. (See Appendix A for a system model example.) The proposal also contains a number of further structural improvements, such as inheritance between models and so called property links, i.e. link types that have associated media models.
Other primitive model types	Another area of development which is pointed out in the NMF report is to include e.g. causal, algorithmically described models that operate in discrete time. This type of model is needed primarily to express discrete time controllers. A number of other interesting model categories have also been envisioned.

Model documentation Significant efforts have been made in the area of representing additional knowledge about a model. Some of this will be possible to formalize, other portions will be best represented as structured text. Contributors in this field are e.g. [Pelletret 1995].

Liaison with STEP General product modelling (PM) is an important related field. It should be natural in the future to also express model behavior in a PM. Indeed, PMs of many types of objects, e.g., controllers, are rather meaningless without any account of object behavior. NMF and similar languages are obvious candidates for this type of description. A discussion of the relationship between NMF and STEP-related work can be found in [Sahlin and Johansson 1994].

3. Basic Constructs

3.1 A Simple Example - Thermal Conductance

The best way to get to know NMF is to examine models. We suggest that you, as soon as the basics are becoming clear, study some of the models that are shipped with the ASHRAE Translator. The models in Appendix 2 of the NMF report provide some additional material, but many of them have been selected to illustrate various features and are relatively advanced. Here we will use a very simple model, a thermal conductance, to illustrate the basic NMF constructs and features. Take a moment to regard the model, we will comment on the details in the following sections.

Figure 3-1. A thermal conductance with a linear relationship between heatflux, Q, and temperature difference, T1 - T2.

```
CONTINUOUS_MODEL     tq_conductance

ABSTRACT    "Linear thermal conductance"

EQUATIONS

   /* heat balance */

 0 = - Q + a_u * (T1 - T2);

LINKS

    /* type       name             variables... */
         TQ      terminal_1        T1,   POS_IN Q ;
         TQ      terminal_2        T2,   POS_OUT Q ;

VARIABLES
   /* type        name         role       description */
     Temp         T1           IN         "1st temp"
     Temp         T2           IN         "2nd temp"
     HeatFlux     Q            OUT        "flow from 1 to 2"

PARAMETERS
   /* type        name         role       description */
     Area         a            S_P        "cross section area"
     HeatCondA    u            S_P        "heat transfer coeff"
     HeatCond     a_u          C_P        "a * u"

PARAMETER_PROCESSING
     a_u := a * u;

END_MODEL
```

NMF Model 3-1: tq_conductance

3.2 What is a continuous model?

A continuous model operates in continuous time. In NMF version 3.X, this is the only model type available. Future versions of NMF will also encompass discrete time models, i.e. models which operate with difference equations and (usually) a fixed timestep, and other model categories. In continuous models, relationships between model variables are expressed in terms of equations, equalities, that from the modeller's perspective can be regarded as being fulfilled at all times. The fact that most simulation environments use a finite timestep for the actual calculation should generally be disregarded when constructing NMF models.

NMF continuous models may handle discontinuities in functions and driving data. This is treated in Section 5.4 Multiple Modes and Discrete Events. Event *driven* models, where *all* dynamics are the result of sudden "happenings" are not on the immediate NMF agenda. Such models are often used for simulation of man-made systems, such as a traffic situation or the logistics of a manufacturing plant.

3.3 Nomenclature: Comments, Reserved Words, etc.

NMF comments are delimited by `/*` and `*/` . They may contain multiple lines and may occur anywhere in the source (except inside tokens).

NMF identifiers, i.e. names of variables, parameters, links etc., must not exceed 31 characters in length. They must start with a letter, but may also contain digits, underscore '_' and dollar signs '$'.

Case has no formal meaning in NMF, but conventionally reserved words and global constants are written with UPPERCASE. Variables are Capitalized and parameters are lowercase.

3.4 Abstract Section

The abstract should be a brief text, usually with multiple lines, delimited by double quotes: `"multiline text"`.

3.5 Equation Declarations

The relationships between model variables and parameters are expressed by stating algebraic and/or ordinary differential equations in the `EQUATION` section. In Example 3-1, Ohm's law is stated for the relationship between the three variables, `Q`, `T1`, `T2`, and the parameter `a_u`. Note that the statement of this equation has no implication on the question what variable to "solve" for. All three variables have equal status. We are simply stating a relationship between them that is valid at all times.

Several linear or non-linear equations may be stated in an NMF model. Their individual order is of no consequence to the meaning of the model or to the generated solution algorithm. This is generally a bit confusing to NMF beginners that are accustomed to express models in terms of executable solution sequences, and it requires some time to get used to. However, once the concept has been grasped, most users find themselves able to concentrate on real modelling issues rather than on numerical solution methods.

Equations may refer to any Fortran 77 floating point function or to any user defined function, which may be written in Fortran, C, or directly with NMF assignment constructs. Declaration of user defined functions is treated in Section 5.2 User Defined Functions. A special construct for piecewise defined functions is provided, the *conditional expression*:

```
c*T' = IF x < 0 THEN  0
       ELSE_IF x > 2 THEN  Q2
       ELSE_IF x > 1 THEN  Q1
       ELSE  Q0
       END_IF;
```

First order ordinary differential equations (ODEs) are declared by appending an apostrophe character (ASCII code 027h) after the name of the time-differentiated variable. To express higher order derivatives, intermediate variables must be introduced. The following equations illustrate declaration of ODEs and usage of a user defined function `g`:

```
V = X';           /* introduce intermediate variable V */
g(m, V',F) = 0;   /* g could e.g. evaluate (m*V' - F)
                     so we get   m*X'' = F */
```

Time in NMF models is always measured in seconds, i.e. derivatives imply differentiation with respect to time in seconds. Some target environments measure time in other units, and the model code generated by the translator will automatically introduce a proper constant to convert to the time unit prescribed by the target environment.

3.6 Model boundaries - LINKS

In the `LINKS` section, the communication ports of the model are specified. The thermal conductance in Example 3-1 has two links, `terminal_1` and `terminal_2`. Each link has two variables, a temperature and a heatflux. *Only variables appearing in a link statement may interact with other models*. This encapsulation of component internal behavior is an important model structuring principle, which will be further discussed.

Links are typed globally. The links in the example are of the `TQ` type, which is also referred to by many other continuous models. The `TQ` links must always have a temperature in the first position and a heatflux in the second. Global types and their declaration are discussed in Section 3.9 Global Declarations.

Flow type variables (further discussed in Section 3.9 Global Declarations) are specified in terms of positive direction in the `LINK` statement. In the example, the heatflux `Q` is defined to be positive in the direction from `terminal_1` to `terminal_2`, i.e. it is positive into `(POS_IN)` the first terminal and positive out `(POS_OUT)` of the second.

Links must always be specified in an NMF model, but they are not used by all target environments. In link supporting environments, the variables in two connected links are joined collectively. Suppose, for example, that we have two `tq_conduction` instances called *Cond_1* and *Cond_2* and that their respective `terminal_2`:s have been joined.

In current NMF (version 3.02) there are no constructions for system modelling, as would be required for this example. However, the next generation NMF (version 4) is well underway and the (formally approved) syntax for joining two (previously declared) model instances is the following. A full example of a system model in NMF v. 4 can be found in Appendix A.

```
//NMF version 4 syntax example

CONNECTIONS     /*Link level internal connections*/

/*   instance.link     = instance.link; */

     Cond_1.terminal_2 = Cond_1.terminal_2;
```

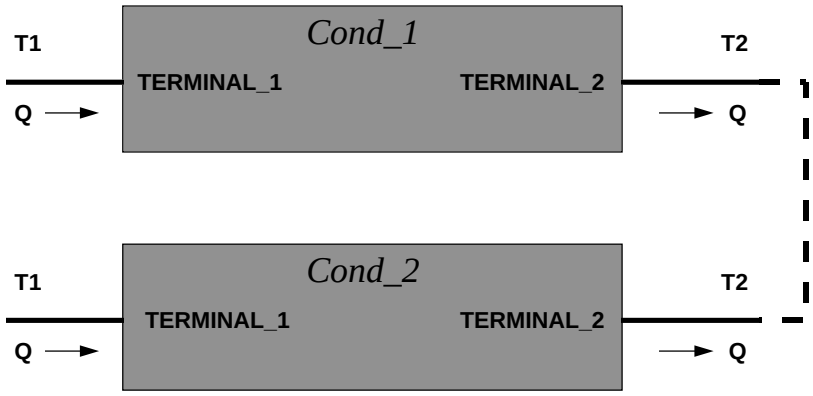

Figure 3-2. Two connected tq_conduction instances

This results in the generation of the following connection equations in the target environment (using dot notation here to separate variable identities):

Cond_1.T2 = Cond_2.T2,
Cond_1.Q = - Cond_2.Q

Note the minus sign in the second equation, that is due to non-matching positive directions. Both instances have `Q` as positive out of `terminal_2`.

3.7 Variable and Parameter Declarations

Variables and parameters are explicitly declared in an NMF model. Parameters are quantities that remain constant throughout every simulation. Each *quantity*, a variable or parameter, must be declared in four respects:

3.7.1 Type

Similarly as for links, variables and parameters, are globally typed. As an alternative, they may be declared GENERIC, which roughly means that they are compatible with any other type.

3.7.2 Name

The local name for the quantity.

3.7.3 Role

Variables are divided into four different roles: `IN`, `OUT`, `LOC`, and `A_S`. The two latter concern assignment modelling and are treated in Section 5.3 NMF Variable Assignments. A modeller is required to specify one possible selection of given (`IN`) and calculated (`OUT`) variables, and *the number of `OUT`-variables must be equal to the number of equations in the model*. The selection should be made to maximize model robustness. The reasons for requiring this information are discussed below.

Parameters can be either supplied, `S_P`, or computed, `C_P`. The former are given explicitly by the user, while computed parameters are calculated in the `PARAMETER PROCESSING` section, which is executed once, prior to the actual simulation. Both `S_P` and `C_P` parameters may occur in equations in the `EQUATION` section.

3.7.4 Description

A descriptive string must be given for each variable or parameter. It cannot exceed 80 characters. Each white space between words is counted as a single character, even if it in fact contains several tab, newline, or space characters.

In addition to this, a quantity may optionally be given a default value and an interval in which the model is valid. These declarations are described further in the NMF report, Sections 4.2.4 - 5.

3.7.5 IN/OUT Discussion

The explicit selection of given (`IN`) and calculated (`OUT`) variables for a model may seem to contradict the principles of equation-based input-output free modelling that form the basis of NMF. However, for environments where models are packaged as subroutines, with inputs and outputs, a selection is needed. The `IN`/`OUT` labels on variables in the NMF source code is one way to signal the desired partitioning. The ASHRAE translator to TRNSYS and HVACSIM+ utilizes this method.

Consequently, one reason for including this information is that maximal compatibility is attained with the large number of existing input-output oriented environments. Using the `IN/OUT` information, it is always possible to translate an NMF model into these environments directly, without adding anything extra. It should also be observed that most models in the libraries of such environments exist in a single input-output version.

For many application domains, only a few models are duplicated with different input-output structures to attain connectivity. A further discussion of these issues can be found in Section 3.1 of the NMF report. The issue is also discussed in Exercise 1.

The more fundamental argument for the `IN-OUT` partitioning is that different selections have different robustness properties, and this is taken advantage of in some environments. The given selection should be a "good inverse" for the whole model (in the matrix sense). An extreme example is a limited proportional controller, where it is possible to calculate the in signal from the out signal in the proportional band, but not in any saturated state. See also Section 3.5 of the NMF Report.

3.7.6 Order of Declared Quantities

The order of quantity declarations does not influence the mathematical meaning of a model. However, an NMF implementation is allowed to use the order for other purposes. Frequently, the order of variables and parameters in the generated code is identical to that of the NMF source. In the ASHRAE Translator, for example, the order of declarations is used to generate an order for the input, output, and parameter vectors of the TYPE subroutines. (See the Generated Code Section of the user's manual.)

3.8 Parameter Processing

In the `PARAMETER_PROCESSING` section, computed parameters (`C_P`) are calculated from supplied ditto (`S_P`). The code is executed once at the start of a simulation. A limited range of algorithmic constructs such as `IF <condition> THEN ... ELSE ... END_IF` are available. There are no potentially endless iteration constructs such as `WHILE <condition> DO`.

Standard and user-defined functions may be referred to, as in the `EQUATION` section. Standard functions include all Fortran 77 floating point operations, and a routine for signaling errors `NMF_ERROR`, which takes any number of strings and expressions as arguments. `NMF_ERROR` and other special functions are discussed in Section 4.3 of the NMF Report.

```
IF Re < 2500 THEN
   CALL  NMF_ERROR ("Laminar flow, Reynolds number = ", Re)
END_IF;
```

3.8.1 Restriction on Repeated Assignments

Compared to regular programming, one important difference applies. *In NMF, a quantity may not be assigned to repeatedly*. It may or may not be assigned to - depending on the thread of execution - but once assigned, it is illegal to update the variable again.

The reason for this limitation is that it should, for the benefit of pure equation based environments, always be possible to interpret NMF assignments as if they were equations. The limitation also enables a range of symbolic processing constructs that would otherwise be out of reach.

One consequence of this restriction is illustrated by the following example.

```
FOR i = 1, n                          /* NOT permitted */
  help := a[i] - b[i] ;               /* help assigned repeatedly */
  c[i] := IF help < 0 THEN
            0
          ELSE
            help**2
          END_IF ;
END_FOR ;
```

Two alternative ways to respect the restriction are shown:

```
FOR i = 1, n
  c[i] := IF a[i] - b[i]  < 0 THEN
            0
          ELSE
            (a[i] - b[i])**2
          END_IF ;
END_FOR ;
```

```
FOR i = 1, n
  help[i] := a[i] - b[i] ;
  c[i] := IF help[i] < 0 THEN
            0
          ELSE
            help[i]**2
          END_IF ;
END_FOR ;
```

Another aspect of the restriction relates to conditional assignments. Here, the interpretation of the restriction is less obvious.

In the present version of NMF, it is formally permitted to lexically assign a variable in all of a series of mutually exclusive `IF THEN ... END_IF` constructs. This possibility will most likely be removed in future versions, and it is therefore recommended to assign a parameter only within a single, possibly complex, conditional construct.

This construction, for example, is presently formally allowed but not recommended:

```
IF test <= 1 THEN
  a := 2
END_IF;
...
<other assignments>
...
IF test > 3 THEN
  a := 4
END_IF;
```

A recommended construction is:

```
IF test <= 1 THEN
    a := 2
ELSE_IF test > 3 THEN
    a := 4
END_IF;
```

Note that in both these constructions, a will not always be defined.

3.9 Global Declarations

The basic idea with NMF is to provide a way to exchange models between developers. One thing that makes model exchange difficult today is that people naturally make different choices in rather trivial matters such as selection of units and variables. Some prefer to use e.g. enthalpy to get compact equations, others pick temperature to gain engineering appeal. The modest ambition of NMF is to recommend some choices, to attain standardization on a voluntary basis, but without limiting the freedom of the individual modeller.

Global declarations include *quantity types*, ordered lists of quantity types called *link types*, and global *constants*. Global declarations is sometimes also called *foundation*. A basic list of recommended global declarations will be issued regularly by the NMF Committee. A user may add to or change this list as necessary. An excerpt from the present `global.nmf` file is:

```
QUANTITY_TYPES

/* type name      unit              kind */

   Area           "m2"              CROSS
   Control        "dimless"         CROSS
   Density        "kg/m3"           CROSS
   Factor         "dimless"         CROSS
   Force          "N"               CROSS
   HeatCap        "J/(K)"           CROSS
   HeatCapA       "J/(K m2)"        CROSS
   HeatCapM       "J/(kg K)"        CROSS
   HeatCond       "W/K"             THRU
   HeatCondL      "W/(m K)"         THRU
   HeatCondA      "W/(m2 K)"        THRU
   HeatFlux       "W"               THRU
   HeatFlux_k     "kW"              THRU
   Length         "m"               CROSS
   MassFlow       "kg/s"            THRU
   Pressure       "Pa"              CROSS
   Temp           "Deg-C"           CROSS

LINK_TYPES

/* type name       variable types... */

/* generic        (arbitrary, arbitrary,...) implicitly defined */

   Q              (HeatFlux)
   T              (Temp)
   TQ             (Temp, HeatFlux)
   PMT            (Pressure, MassFlow, Temp)
```

```
   PMTQ            (Pressure, MassFlow, Temp, HeatFlux)
                   /*PMT(Q) may be used for any fluid*/

   BidirAir        (Pressure, MassFlow, Temp, HeatFlux)
                   /* BidirAir should be used for air only*/

   ControlLink     (Control)

CONSTANTS

  /* name       value       unit             comment */

   ABS_ZERO     -273.16     "Deg-C"        /* absolute zero temp */
   BOLTZ        5.67E-8     "W/(m2 K4)"    /* Stefan Boltzmann */
   G            9.81        "m/s2"         /* gravity acceleration */
   GASCON       287.        ""             /* general gas constant */
   HF_VAP       2.501E6     "J/kg"         /* water vaporization heat */
   P_ATM_0      1.013E5     "Pa"           /* standard air pressure */
   PI           3.1415927   "dimless"      /* the pi number */
```

In NMF models, the global declarations are referred to by variables and parameters (quantity types), by links (link types), and directly in equations or assignments (constants). For instance, in the `tq_conduction` model, the `TQ` link type, the `Temp` and `Area` quantity types are used in this way. No global constants are used in the `tq_conduction` model.

3.9.1 Name

Naming of declared quantities (identifiers) should adhere to some guidelines that are specified in Appendix 3.2 of the NMF report. See also Section 3.3 Nomenclature: Comments, Reserved Words, etc. of this text.

3.9.2 Unit

The unit is presently given as a text string, the structure of which is not formally specified. It is however recommended to adhere to the present style, since automatic parsing of unit information is a natural NMF extension that most likely will be handled by future translators. So, use the present style of writing units to make your models compatible with future automatic unit checking.

3.9.3 Kind

The kind flag should be `THRU` for flow type variables, and `CROSS` for all others. `THRU` variables are fluxes with a direction associated with them. The direction of a `THRU` variable must be specified if the variable appears in a `LINK` statement in a continuous model, as positive into or out of the current model. When two `THRU` variables from different models are connected, the connecting equation will have a minus sign in it, if the defined flow directions do not match, i.e. Kirchoff's law is applied to the "node" between the connected models. `CROSS` variables on the other hand are always set equal to each other.

A general piece of advice is that any variable that you do not immediately classify as being of flow type, should be `CROSS`. It would be possible to define a modelling language with only `CROSS` variables. Indeed, languages exist where it is up to the user to make sure that implicit definitions of positive direction (in equations) always match. NMF offers extra service in this respect. For thermodynamical systems the service is

very valuable, for some other applications it is less useful. (See for example Exercise 1.)

Tricky considerations do occur. Imagine, for example, that we would like to *monitor* the flow through our `tq_conduction`. We then need a link, let us call it `measure`, that allows the monitoring component, perhaps some controller, to read the present flux. However, we want to associate a direction only to the flux itself, and not to the *measurement* of the flux, which should be a `CROSS` variable. One solution could be to define a global type

```
  HeatFluxCross   "W"   CROSS
```

and to add the following declarations to the `tq_conduction`

```
...
EQUATIONS
....
/* monitor flow*/
  0 = -Q_m + Q;

LINKS
....
  GENERIC   measure   Q_m;

VARIABLES
....

HeatFluxCross  Q_m  OUT  "measured heat flux"
....
```

Here we have typed the `Q_m` variable explicitly, to illustrate our point, but use the `GENERIC` type for the `measure` link. We could have introduced a global type for this specific single variable link type as well.

3.9.4 The `GENERIC` Reserved Word

The `GENERIC` type can always be used if one wishes to avoid type checking, both for links and for variables and parameters. The following rules apply:

1. A `GENERIC` variable is always of `CROSS` type

2. A `GENERIC` variable can occur in any `CROSS` position in a typed link

3. A `GENERIC` link can have any number of variables of any type. It can be connected to any link of matching length. Type checking is then done at the variable level.

4. A `GENERIC` variable may be connected to any `CROSS` variable

In the example above, it would generally have been better style to avoid the introduction of the rather esoteric `HeatFluxCross` type and use `GENERIC` instead.

3.10 File Structure

NMF does not specify the precise structure for source files. Global declarations must however be kept together in a single file, possibly decomposed into several include files. Some implementations have each model in a single file, others have all models that belong to the same (informal) model family in a single file. The ASHRAE translator can effectively handle both these file organizations.

4. Modelling Guidelines

In this section we will attempt to convey some qualitative aspects of the art of NMF modelling. Experience has been gained from several significant simulation projects, and some general conclusions regarding modelling methodology can be drawn.

The set of basic constructs that were presented in the previous section represent the core that is needed for simple models. In this section we will base the discussion on these basic constructs. The more advanced constructs that will be treated later will not affect the validity of the modelling guidelines presented here.

We will introduce some general guidelines in the first sections and then illustrate them with an example regarding pressure-flow modelling with models which allow massflow to change direction.

4.1 Thinking Equation Declaration - Not Assignment Programming

The difference between an algorithmic description of a model and an equation description is frequently misunderstood. NMF is sometimes erroneously thought of as a proposed standard for algorithmic descriptions. Let us look for a moment at the difference between algorithms and equations.

4.1.1 Algorithmic Models

An algorithm is used to describe a computation procedure, a step by step recipe on how to get from input to output. Some important characteristics of algorithms are:

1. Variables are defined as either input, output, or local
2. A variable receives it's value by assignment of the value of some computable expression, possibly involving the variable itself
3. The same variable may be assigned to repeatedly
4. Variables are often repeatedly assigned in loops, which are executed until some criterion is met, i.e. not an easily predictable number of times.

4.1.2 Equation Models

Equation models state a mathematical relationship between variables. The following characteristics apply:

1. If the number of variables, *k*, exceeds the number of linearly independent equations, *N*, then $k - N$ variables may be given as input, and the remainder, *N*, can usually be solved for analytically or numerically.
2. A variable should be thought of as having a value at all times; the equations may be more or less satisfied depending on current variable values. If anything is to be called *output* of the equation model, it is the value of the residuals in each equation.
3. The order of equations and the arrangement of terms in equations is irrelevant to the meaning of the model.

4.1.3 Discussion

The main advantage of expressing models in terms of equations rather than as algorithms is that an equation model generally can be *automatically* converted into an algorithm, whereas the opposite is only possible in a small number of cases. Furthermore, several algorithms with, for example, different input-output configurations can be generated from the same equation model. Equation models also lend themselves to other types of symbolic processing, which is of great importance to simulation, for example automatic differentiation.

Many people with programming experience tend to look for the same concepts and tools for NMF work as they are accustomed to. Some things are certainly common for all types of formal coding work, but one must be aware of the differences. Some examples are:

1. It is generally misleading to think of a continuous model as a concept similar to a subroutine or a function in a regular language. A better mental picture is to think of the model as a class definition in an object oriented language. A class which can be instantiated multiple times, each with its own private set of data, and each with the ability to say how well it's equations are fulfilled.
2. There is far less need for sophisticated data structures in the NMF domain than in a regular language.
3. NMF model definitions tend to be reused in various configurations and by other users to a greater extent than what is generally the case for subroutine and class definitions.
4. NMF modelling is more work intensive than most other regular programming, in terms of produced code per time unit.

4.2 Finding Component Model Boundaries

The first task to tackle in developing a library of NMF models for a specific application is to divide the system to be simulated into separate component models. The real art of NMF modelling lies in finding the proper boundaries for component models. Many physical systems have a natural decomposition into component models, that one should try to mimic in the modelling. Others, such as a thermal zone model, require much more careful planning in order to achieve a good structure with versatile component pieces. Some things to consider are:

1. Identify "flow circuits" in the system, where a common link type can be used to communicate between individual components. Try to keep the number of link types to a minimum. Examples of such circuits in a building simulation are: the duct system, the water pipe system, heat transfer between objects (using, for example, the `TQ` link type).
2. Define component boundaries that minimize the data shared between components, i.e. try as far as possible to avoid cases when two models need the same variable or parameter.

3. There is no optimal size of a component model, large models are sometimes more practical, small usually more versatile. For most solvers, large models are more computationally efficient.

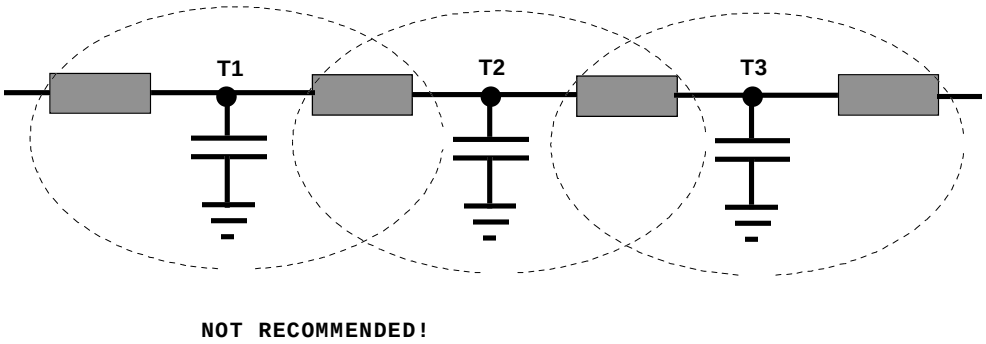

Figure 4-1. Dashed boundaries are a possible subdivision of an RC-network into NMF component models with only a single variable, T, on the links. The disadvantages are that the resistor (or conductor) parameters are duplicated in neighboring models, and the equation for ohm's law is similarly duplicated.

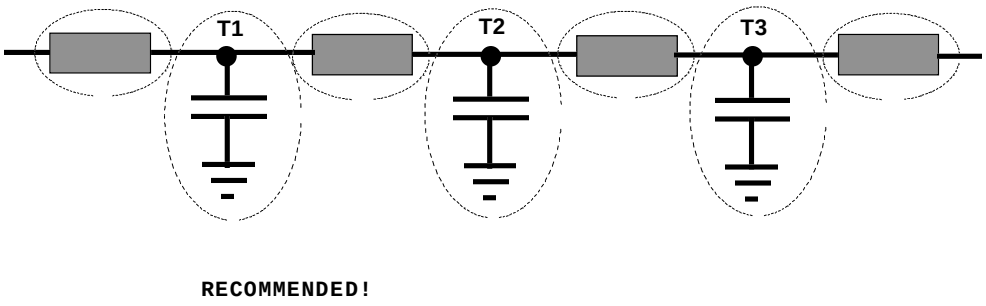

Figure 4-2. A component subdivision with better encapsulation properties, no parameters or variables are shared. The cost is an extra link variable, Q, for the flow between components. This is in most cases a preferred architecture, unless the extra cost in additional boundary variables is unacceptable.

Take a look at some component model libraries that are shipped with the translator to get a feel for model architecture. Some libraries have been translated from other sources and they may not be ideal from an NMF architectural point of view. Most models contain advanced constructs, as will be presented in the next section, but the architectural concepts remain the same.

4.3 The Right Number of Equations in Each Model

It is sometimes difficult to decide what equations go where. Two general guidelines apply:

1. Try to make models as self-contained as possible.

2. Say everything there is to say about a component, but only once. Never state the same equation twice.

One must rely on "engineering intuition" here, but this seldom leads to any significant problems in practice.

4.4 A Useful Special Case - Bi-directional Flow

We will now turn to an example rooted in a real life NMF application, multizone airflow.

The Multizone Air-Exchange (MAE) models constitute an example of a component family that have been developed primarily for IDA, but that should be useful in other environments as well. However, these models are rather numerically demanding, and it is not reasonable to expect that they will perform satisfactorily with any solver. The issue of limitations in the matching between models and solvers will be discussed further in Section 4.7 Target environment capabilities.

The MAE models, in their basic form, predict pressure levels, airflow and transport of convected energy in a network of zones and leaks between zones and in the associated ventilation system. In the version, used here in the example, a room as well as a junction in the ventilation system is modeled as a well mixed zone. This section deals with the basic equations of the MAE models and describes the corresponding NMF code for two of them.

4.4.1 The Multizone Air-Exchange Models

In the MAE models, macroscopic equations are used, assuming complete mixing within each zone (node). In their most elementary versions, the models include the basic pressure - mass balance and transport of enthalpy.

There are two basic groups of components; nodes and connecting elements. The node components are characterized by their potentials, i.e., pressure (P) and temperature (T). Nodes can be zones, or junctions in the ventilation system. Their model equations represent conservation of mass and energy:

$$0 = \sum m_i \qquad (1)$$

$$0 = \sum q_i + q_{zone} \qquad (2)$$

where q_{zone} is a heat source, possibly from energy transfer with the envelope.

The connecting elements can be leaks, or any double-ended pieces of the ductwork (ducts, grilles, fans, etc.). Among the model equations of a connecting element there is always the mass flow modeled as a function of the pressure difference over the element:

$$m_{1\text{-}2} = f\,(P_1 - P_2) \qquad (3)$$

In the example of this text power law equations are used for the pressure - flow relation. For a leak this implies:

$$f(\Delta p) = \begin{cases} c(\Delta p)^n, & \Delta p > 0 \\ -c(-\Delta p)^n, & \Delta p < 0 \end{cases} \qquad (4)$$

where n is a coefficient between 0.5 and 1.0, depending on flow type, and c is a "conductivity" coefficient.

Heat (enthalpy) transport through a connecting element is convected by the mass flow:

$$q_{1\text{-}2} = \begin{cases} c_p T_1 m_{1\text{-}2}, & m_{1\text{-}2} > 0 \\ c_p T_2 m_{1\text{-}2}, & m_{1\text{-}2} < 0 \end{cases} \qquad (5)$$

where T_1 and T_2 are the temperatures of the nodes connected by the element.

This type of modular approach makes it easy to replace individual component models as long as the interface variables between models are the same. Thus, it is possible to refine on the MAE models by replacing the above equations by more detailed ones.

4.4.2 NMF Examples

In order to illustrate some basic features two examples of NMF code are shown below. The NMF code in the first example models a zone with three links and the next a leak between zones.

The link type, `BidirAir`, handles air-exchange interaction in terms of bi-directional flow between neighboring models. It has four variables:

```
BidirAir       (Pressure, MassFlow, Temp, HeatFlux)
```

The zone model also has a `TQ` link. It is an interface for temperature and heat flow between the air-exchange family of models and the thermal models for the building envelope.

```
CONTINUOUS_MODEL  simple_bdzone

ABSTRACT
  "A static zone model for air-exchange modelling. Bidirectional
   transports of energy is modelled."

EQUATIONS

/* mass conservation (eqn (1))*/

   0 = M_0 + M_1 + M_2;

/* energy conservation (eqn (2))*/

   0 = Q_zone + Q_0 + Q_1 + Q_2;

LINKS

  /* type     name        variables... */

BidirAir    terminal_0    P, POS_IN M_0, T, POS_IN Q_0;
BidirAir    terminal_1    P, POS_IN M_1, T, POS_IN Q_1;
BidirAir    terminal_2    P, POS_IN M_2, T, POS_IN Q_2;

Tq          air_temp      T, POS_IN Q_zone;

VARIABLES

  /* type      name   role [def   min  max]  description */

  MassFlow     M_0    OUT                "terminal 0 massflow"
  MassFlow     M_1    IN                 "terminal 1 massflow"
  MassFlow     M_2    IN                 "terminal 2 massflow"
  Pressure     P      IN                 "zone pressure"
  HeatFlux     Q_0    OUT                "terminal 0 HeatFlux"
  HeatFlux     Q_1    IN                 "terminal 1 HeatFlux"
  HeatFlux     Q_2    IN                 "terminal 2 HeatFlux"
  Temp         T      IN                 "zone temperature"
  HeatFlux     Q_zone IN                 "heat gain/loss in zone"

END_MODEL
```

NMF Model 4-1: simple_bdzone

As you can see, the zone has a pressure and a temperature that is implicitly defined by neighboring components. These variables do not occur explicitly in the zone equations, but only on the links.

```
CONTINUOUS_MODEL simple_bdleak

ABSTRACT "A simplified powerlaw leak model w/ bidirectional
transports tempered air."

EQUATIONS

/*driving pressure difference*/

  Dp = P1 - P2;

/* powerlaw massflow equation (eqns (3) and (4))*/

  M = IF Dp > 0 THEN c * Dp**n
      ELSE -c * (-Dp)**n
      END_IF ;

/* convected heat through leak (eqn (5))*/

  Q = IF M > 0.0 THEN cp * T1 * M
      ELSE cp * T2 * M
      END_IF ;

LINKS

/* type      name        variables... */

BidirAir    terminal_1   P1, POS_IN  M, T1, POS_IN  Q
BidirAir    terminal_2   P2, POS_OUT M, T2, POS_OUT Q;


VARIABLES

/* type    name  role def  min  max  description */

MassFlow     M    OUT  0   -BIG    BIG  "massflow through leak"
Pressure     P1   IN   1   -BIG    BIG  "terminal 1 pressure"
Pressure     P2   IN   2   -BIG    BIG  "terminal 2 pressure"
Temp         T1   IN   20 ABS_ZERO BIG  "Temperature of neighbor 1"
Temp         T2   IN   20 ABS_ZERO BIG  "Temperature of neighbor 2"
HeatFlux     Q    OUT  0   -BIG    BIG  "heat moved by massflow"
Pressure     Dp   OUT                   "effective pressure diff."

PARAMETERS

/* type   name  role def  min  max  description */
Generic   c     S_P  1    0     BIG "powerlaw coeff. [kg/(s Pa**n)]"
Generic   n     S_P .5   .5     1.0 "powerlaw exponent [dimless]"
HeatCapM  cp    S_P  1006 .5E3  3E3 "air cp"
END_MODEL
```

NMF Model 4-2: simple_bdleak

These two models can be combined into arbitrary networks. The connection rules are that two zones always must be connected via one or more leaks. Leaks may be connected in series. These rules are not formalized in the models, but are implied by the way the equations have been formulated. In future versions of NMF, connection rules like these may well be formalized.

Key to the understanding of these models is the fact that the pressure and temperature that appear in the leak model are valid for the *neighboring* components. `P` and `T` of the zone model become computable since they occur in equations in the leak model.

A new concept in the leak model is the default, minimum, and maximum values that may be defined for each variable or parameter. Most real models have such explicit limits coded.

The zone and leak models have the `BidirAir` link allowing for bi-directional flow. Let us model a ventilation component with unidirectional flow as well, since bi-directional flow modelling in the ventilation system may be too costly (more variables and equations) for some applications.

For unidirectional modelling we need only three link variables: pressure, massflow and temperature, and, generally, only a single equation (3). The three-variable link type is called `PMT`. Simple link types are generally named after the variables in them, whereas complex ones have other descriptive names.

We will present a single unidirectional model, a supply terminal model, `simple_VxSupT`, that acts as interface between zone models, with `BidirAir` links, and the supply side of the ventilation system, with `PMT` links. Consequently, the `simple_VxSupT` model has one link of each kind. Although only uni-directional flow is allowed, it still needs two equations (in addition to the calculation of `dp`) since there is a `BidirAir` link present.

```
CONTINUOUS_MODEL simple_VxSupT

ABSTRACT "Supply Terminal. Interface between unidirectional
supply system and bidirectional links in zones. Power law
pressure drop"

EQUATIONS

  Dp = P1 - P2;

/* powerlaw massflow equation */

  M = c_t * sqrt (Dp);

/* convected heat through terminal */

  Q = cp * T1 * M;

LINKS

  /* type      name        variables... */

  PMT          inlet       P1, POS_IN  M, T1;
  BidirAir     zone        P2, POS_OUT M, T2, POS_OUT Q;


VARIABLES

/* type        name    role def   min  max   description */

  massflow     M       OUT  0.    0    BIG   "massflow through leak"
  Pressure     P1      IN   2.  SMALL  BIG   "pressure in"
  Pressure     P2      IN   1.  SMALL  BIG   "pressure out"
  temp         T1      IN   15. ABS_ZERO BIG "temperature in"
  temp         T2      IN   15. ABS_ZERO BIG "temperature in zone"
  HeatFlux     Q       OUT  0.  -BIG   BIG   "heat convected by
                                              massflow"
  Pressure     Dp      OUT  1     0    BIG   "pressure drop"

PARAMETERS

  /* type   name  role  def    min   max  description */
  generic   c_t   S_P    0.     0   BIG   "powerlaw coefficient"
  HeatCapM  cp    S_P   1006   500  3000  "air cp"
END_MODEL
```

NMF Model 4-3: simple_VxSupT

4.5 Real-Life Equations

The MAE models are highly non-linear and therefore rather demanding to solve numerically. For non-linear problems a solver must start with an initial guess of the solution. The initial guess is generally supplied by the user. Given this guess, the solver will try to improve the solution using various schemes. The quality of such numerical schemes is determined by their ability to safely produce a solution given a poor initial guess.

In the solution process, the solver will evaluate the system of equations in a number of tentative points in the variable space. The location of these points can generally not be predicted at the modelling stage. An important model quality characteristic is therefore

that models permit evaluation also in odd variable combinations, without risking division by zero or that the derivative of an equation with respect to its variables becomes infinite. Most solvers try to estimate such derivatives in the solution process.

The simple MAE models that we have just discussed, will work well for good quality initial guesses, if the massflows in the solution process stay away from zero. One problem is that the derivative of equation (4) with respect to Δp becomes infinite around zero, and since the purpose of the models is to predict bi-directional flow, they should be robust for evaluation in the neighborhood of zero massflow.

Solution ideas to these types of difficulties can often be obtained from the physics of the problem. In our case, for a small enough flow through an opening, there will be a switch to laminar flow, with a corresponding linear relationship between flow and pressure. If equation (4) is augmented with a linear section around zero, we should be able to avoid the problem:

$$f(\Delta p) = \begin{cases} c(\Delta p)^n, & \Delta p > \Delta p_0 \\ c_0 \Delta p, & \text{abs}(\Delta p) < \Delta p_0 \\ -c(-\Delta p)^n, & \Delta p < -\Delta p_0 \end{cases} \qquad (4b)$$

where the parameter Δp_0 could be selected to be smaller than any interesting pressure difference to resolve. c_0 can be computed to give a continuous transition between the different regimes of equation (4b).

Explicit isolation of singularities like we have just seen, and equations that are possible to evaluate in any "strange" regime are characteristics of a good quality model. A classical pitfall is when a polynomial is used to interpolate between measured points on some curve, e.g. a fan curve. The polynomial fit may work reasonably well within the operating regime of the fan, but can be completely off in other areas, even having the wrong sign. A solver is much more likely to converge if the fan curve has a reasonable slope outside of the (physical) operating regime as well.

4.5.1 User-Independent Initial Guesses

Another problem with the MAE models, as with any non-linear model, is that they require a reasonable initial guess to be provided in order to converge safely. This may be acceptable for some applications, and equally unacceptable for others. If for example, a few thousand non-linear equations are solved, it becomes both difficult and tedious to provide any non-trivial initial guess.

NMF provides a way around this problem by a system function called `LINEARIZE`. If a solver supports this feature, a call to `LINEARIZE` will return `true` in the first few iterations. The model can then detect this and behave linearly or close to linearly in this case, and thus provide a user-independent and reasonable initial state, since most solvers will solve a linear system in one iteration independent of the starting point.

The `LINEARIZE` function can also be invoked in several steps, to introduce non-linearities in a model successively. Study the specification in Section 4.3.3 in the NMF report.

Equations (3) - (5) in the "real life" MAE leak model `bdleak`, with both isolation of the singularity around zero and a `LINEARIZE` call, look like:

```
/* powerlaw massflow equation */

  M =       IF LINEARIZE (1) THEN c * Dp
            ELSE_IF abs (Dp) < dp0 THEN c0 * Dp
            ELSE_IF Dp > 0 THEN c * Dp**n
            ELSE -c * (-Dp)**n
            END_IF
  BAD_INVERSES () ;

/* convected heat through leak*/

  Q = IF LINEARIZE (1) THEN (T1 - T2) / 2
      ELSE_IF M > 0.0 THEN cp * T1 * M
      ELSE cp * T2 * M
      END_IF
  GOOD_INVERSES (Q) ;
```

The `BAD_` and `GOOD_INVERSES` declarations are discussed in the Section 4.6 Declaration of Bad Inverses.

The linearized model for the heatflux Q deserves some comment. It makes no effort to depict reality. However, if similar constructs are used in other components, the temperatures reached in the linear stage will be averages between boundary temperatures. Given the weak coupling from temperature to pressure in normal ventilation studies, this should mean realistic temperatures in most cases. Thus, this model will not aggravate the difficulties with the non-linear flow balance.

4.5.2 Additional Features of the Multizone Air Exchange Model Family

In the previous sections, exerpts and simplified versions of the MAE models have been presented. The full library is shipped with the ASHRAE NMF Translator. It is a good reference example. The full models contain several language constructs that will be presented in the section on Advanced Constructs. They also model some additional physics that has not yet been discussed:

Stack effect — Driving forces due to temperature dependent density change are modeled throughout the library. Zones are assumed to be well mixed - no temperature gradients are modeled. An oddity in the design is worth mentioning: The models `utleak`, `utsupt`, and `utexht` are used as "boundary objects" to the environment. They all have the outside air pressure at ground level on their `outside` link. The pressure at the level of the actual device is calculated internally.

Contaminant fraction — In addition to the transport of heat with the air, a fraction of some arbitrary substance is also modeled throughout the circuit. Zones have source terms for fraction production, and all models take proper account of the transport and conservation of the fraction.

The MAE models are discussed in more detail in some separate reports [Sahlin, Bring 1993] [Sahlin, Bring 1995].

4.6 Declaration of Bad Inverses

NMF equations are frequently treated symbolically to generate, for example, various inverses. When the ASHRAE Translator generates Fortran code for TRNSYS and HVACSIM+, equations are whenever possible solved symbolically to calculate `OUT` from `IN` variables. Only in the case when no complete symbolic solution is found, does the translator rely on numerical methods. In most cases, the translator is able to automatically determine what symbolic operations can be done without risk of introducing potential errors (generally division by zero). The user can also explicitly declare potentially hazardous inverses of equations, by adding a comma delimited list of `BAD_INVERSES` to an equation before the terminating semicolon. This feature is naturally most important for symbolic solvers that are unable to make good decisions in this respect on their own.

The user can similarly specify `GOOD_INVERSES` of an equation. This is more interesting for a modern symbolic solver, since this way the user can add knowledge about the model that is difficult to deduce automatically.

4.7 Target environment capabilities

NMF tries to overcome many of the trivial problems in transferring models between different simulation environments. There are however fundamental environment differences that prevent some models to be implemented in certain target environments.

The MAE family is a good example of a rather demanding set of models that are unlikely to converge in, for example, a sequential solver such as the traditional TRNSYS solver. The new simultaneous TRNSYS solver (v. 14) should be able to handle the MAE models.

It would be unreasonable to require that any NMF model can be solved in any target environment, since this would, not only in practice be impossible to ensure, but also limit the incentive for solver improvement that simplified model transfer is likely to bring.

A smaller, more practical problem is that most present NMF models have been implemented primarily for use with input-output free solvers, and that the present selection of `IN` and `OUT` variables may prove inadequate for practical work in other environments. Hopefully, the quality of the libraries will be improved in this respect, as they are more widely used in input-output environments, such as TRNSYS v. 13, and HVACSIM+.

Another similar problem is the different conventions regarding units and variable choices that have evolved in the dedicated model libraries around existing environments. It is an impossible mission for NMF to try to be compatible with all of these, and the only feasible solution seems to be the evolution of a new separate NMF based culture in this respect.

5. Advanced Constructs

The basic constructs that we have presented so far are sufficient for small model libraries that are mainly intended to illustrate the concept. Unfortunately, the academic discussion about modelling languages rarely goes beyond such simple examples. However, real-scale modelling puts entirely new requirements on the language. NMF has a strong set of features that enable non-academic modelling. This side of the language has been prioritized at the expense of other features, such as hierarchical modelling, and other types of structuring support, that are mainly needed for system rather than component modelling. Basic system modelling features have been proposed and are now being refined and tested, but they are not discussed in this text.

5.1 Vectors and Matrices

Few other modelling languages have working implementations of vector and matrix support. NMF is built to enable dynamic changes of field dimensions, i.e. an instantiated NMF model can increase field dimensions interactively. This is very useful for a graphical modelling environment where models have, for example, a dynamic number of links. Take for instance the MAE zone model that was discussed in the previous section. It happened to have three `BidirAir` links, but clearly this is no sacred number. In many applications one would need a larger number of links, and in interactive modelling, one would like this number to be flexible. A more comprehensive zone model is:

```
CONTINUOUS_MODEL  less_simple_Bdzone

ABSTRACT
  "A static zone model for air-exchange modelling. Bidirectional
   transports of energy are modelled."

EQUATIONS

/* mass conservation */
   0 = M_0 + SUM i=1, n M[i] END_SUM
   BAD_INVERSES () ;

/* energy conservation */
   0 = Q_zone + Q_0 + SUM i=1, n Q[i] END_SUM
   BAD_INVERSES () ;

LINKS

  /* type    name          variables... */

BidirAir    terminal_0     P, POS_IN M_0, T, POS_IN Q_0;

FOR i = 1, n
  BidirAir  terminal[i]    P, POS_IN M[i], T, POS_IN Q[i]
END_FOR ;

TQ          air_temp       T, POS_IN Q_zone;

VARIABLES

  /* type      name role [def [min max]] description */
  MassFlow     M_0    OUT                "terminal 0 massflow"
  MassFlow     M[n]   IN                 "terminal i massflow"
```

```
  Pressure     P      IN                  "zone pressure"
  HeatFlux     Q_0    OUT                 "terminal 0 HeatFlux"
  HeatFlux     Q[n]   IN                  "terminal i HeatFlux"
  Temp         T      IN                  "zone temperature"
  HeatFlux     Q_zone IN                  "heat gain/loss in zone"

MODEL_PARAMETERS

  /* type   name  role [def  min  max]  description */
  INT        n    SMP    1    1  BIGINT "Number of links minus one"

END_MODEL
```

NMF Model 5-1: less_simple_Bdzone

The model has been complemented with vector variables for mass and heat flow, and now includes a section `MODEL_PARAMETERS`, where the upper dimension `n` of the new vectors is declared. The lower index is always 1. Dimensions of NMF vectors and matrices are declared as separate (model) parameters, and they always have the type `INT`.

Model parameters can be (have `role`) either Supplied Model Parameters `SMP` or Computed Model Parameters `CMP`. The latter category is computed automatically in the `PARAMETER_PROCESSING` section.

The reason for leaving `M_0` and `Q_0` - and as a consequence terminal_0 - as scalars is that declarations, such as `IN`, `OUT` and `BAD_INVERSES` can only be done with respect to the entire vectors and matrices, i.e. special roles for individual vector elements are not permitted.

```
SUM <counter> = <low limit> , <high limit> <expression> END_SUM
```

The summation construction provides a convenient way of writing sums over vector and matrix expressions. The counter variable need not be declared separately and is valid within the evaluated expression, where it can be used in conditional tests etc. The limits must be expressions of parameters only, not variables. If the `<high limit>` is an expression it must be enclosed within parentheses. A counter variable is always an integer and is always increased by 1. Sums may be nested.

```
FOR <counter> = <low limit> , <high limit> <declarations> END_FOR
```

The `FOR ... END_FOR` construction is used to repeat declarations. In the `less_simple_bdzone` it is used for declaring a vector link. Repeated link declarations are currently subject to some restrictions. Links may only be vectors and not matrices, i.e. nested `FOR` loops are not allowed. The vector index may not be an expression. The lower counter limit must be 1, the upper the full dimension of the vector.

When used in the `EQUATION` section, `FOR` loops may be nested. Any parameter expression may limit the counter, and any expression may be used for vector and matrix indexing.

5.1.1 Model Parameter Declaration and Processing

Model parameters are always positive scalar integers (type `INT`). It is important to remember to declare the minimum value of the model parameter for which the model is valid.

Computed model parameters (`CMP`) may be functions of supplied model parameters (`SMP`) ditto, not of regular parameters. They are computed by assignment in the `PARAMETER_PROCESSING` section.

5.1.2 A PDE Example: 1D Heat Equation

A common Partial Differential Equation in many applications is the heat equation. In this section we will formulate an NMF model for heat diffusion in a one dimensional wall.

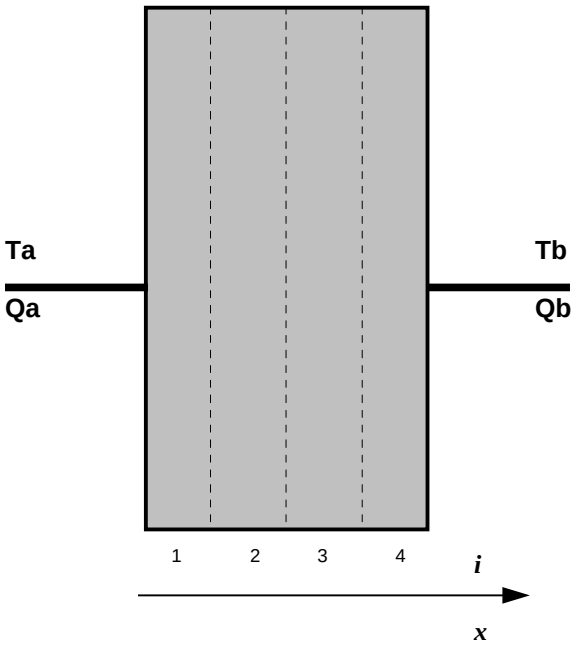

Figure 5-1. A homogeneous wall divided into several layers

$$\frac{\partial T}{\partial t} = \frac{\lambda}{\rho c_p}\frac{\partial^2 T}{\partial x^2} \qquad \text{PDE} \qquad (6)$$

$$Q_a = A\lambda\frac{\partial T}{\partial x}\Big|_{x=0} \qquad \text{BC1} \qquad (7)$$

$$Q_b = -A\lambda\frac{\partial T}{\partial x}\Big|_{x=thick} \qquad \text{BC2} \qquad (8)$$

$$T(x,0) = 0 \qquad \text{IC} \qquad (9)$$

PDE's must be converted into ODE's in order to be expressed with NMF. For the heat equation this means that we must discretize the equation in space but not in time, i.e. we must turn the equation into the following form,

$$\frac{\partial \boldsymbol{T}}{\partial t} = f(\boldsymbol{T}),$$

where $\boldsymbol{T}$ is a vector of discrete temperatures across the wall.

A Taylor series can be used to derive finite difference approximations for continuous differentiation operators of arbitrary order and accuracy,

$$T_{i+1} = T_i + \Delta x \frac{\partial T}{\partial x}\Big|_i + \frac{\Delta x^2}{2!}\frac{\partial^2 T}{\partial x^2}\Big|_i + \frac{\Delta x^3}{3!}\frac{\partial^3 T}{\partial x^3}\Big|_i + \mathrm{O}\left(\Delta x^4\right) \qquad (10)$$

$$T_{i-1} = T_i - \Delta x \frac{\partial T}{\partial x}\Big|_i + \frac{\Delta x^2}{2!}\frac{\partial^2 T}{\partial x^2}\Big|_i - \frac{\Delta x^3}{3!}\frac{\partial^3 T}{\partial x^3}\Big|_i + \mathrm{O}\left(\Delta x^4\right). \qquad (11)$$

Adding equations (10) and (11) allows us to derive the following finite difference approximation (FDA):

$$\frac{\partial^2 T}{\partial x^2}\Big|_i = \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} + \mathrm{O}\left(\Delta x^2\right). \qquad (12)$$

Similarly, subtracting equation (10) from (11) leads to a second order accurate FDA that could be used in the boundary conditions:

$$\frac{\partial T}{\partial x}\Big|_i = \frac{T_{i+1} - T_{i-1}}{2\Delta x} + \mathrm{O}\left(\Delta x^2\right). \qquad (13)$$

In the NMF model below of the wall, two *virtual temperatures*, `Taa` and `Tbb`, have been introduced in order to retain second order (space) accuracy in the boundary conditions. This can be done in other ways as well, e.g. by introducing half-size mesh cells at the boundary. The virtual temperatures can be thought of as representing additional layers of material of thickness `dx` on each side of the actual wall. Since `T[i]` represents the temperature at the center of cell *i*, we must calculate the physical wall temperatures, `Ta` and `Tb`, in separate equations in order to remain second order accurate.

Equation (13) yields a second order accurate estimate of the derivative at the *midpoint* of cell *i*. In the model a second order accurate expression at the cell boundary is required. This can also be derived using Taylor series yielding:

$$\frac{\partial T}{\partial x}\Big|_{i+1/2} = \frac{T_{i+1} - T_i}{\Delta x} + \mathrm{O}\left(\Delta x^2\right). \qquad (14)$$

In the NMF model, the initial state of the temperature vector is only suggested in terms of a common default for the whole vector `T`. Most simulation environments will allow the user to specify an initial state for each individual variable (scalar or vector element) explicitly. There is currently no mechanism in NMF to calculate initial state, in the way parameters may be calculated at the onset of a simulation.

```
CONTINUOUS_MODEL tq_hom_wall

ABSTRACT
   "A 1D finite difference wall model.
    One homogeneous layer.
    TQ interfaces on both sides."

EQUATIONS

/* space discretized heat equation */

   c_coeff * T'[1] = Taa - 2.*T[1] + T[2];
   c_coeff * T'[n] = T[n - 1] - 2. * T[n] + Tbb;

   FOR i = 2, (n-1)
     c_coeff * T'[i] = T[i - 1] - 2. * T[i] + T[i + 1];
   END_FOR ;

/* boundary equations */

   0 = -Ta + .5 * (Taa + T[1]);
   0 = -Tb + .5 * (T[n] + Tbb);
   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  def  min       max    description*/

   Temp     T[n]  OUT   0.   abs_zero  BIG    "temperature profile"
   Temp     Ta    OUT   0.   abs_zero  BIG    "a-side surface temp"
   Temp     Tb    OUT   0.   abs_zero  BIG    "b-side surface temp"
   Temp     Taa   OUT   0.   abs_zero  BIG    "a-side virtual temp"
   Temp     Tbb   OUT   0.   abs_zero  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    role  def  min       max    description  */
   INT       n       SMP   3    3         BIGINT "no of temp layers"

PARAMETERS

/* type      name     role def   min   max  description  */

/* easy access parameters */
   Area      a        S_P  10.   SMALL BIG  "wall area"
   Length    thick    S_P  .25   SMALL BIG  "wall total thickness"
```

```
   HeatCondL lambda   S_P  0.5   SMALL BIG  "heat transfer coeff"
   Density   rho      S_P  2000. SMALL BIG  "wall density"
   HeatCapM  cp       S_P  900.  SMALL BIG  "wall heat capacity"
/* derived 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"

PARAMETER_PROCESSING

    dx := thick / n ;
    c_coeff := rho * cp * dx * dx / lambda;
    d_coeff := lambda * a / dx ;

END_MODEL
```

NMF Model 5-2: tq_hom_wall

5.2 User Defined Functions

NMF supports calls to user defined functions and subroutines. They may be called from the `EQUATION` and `PARAMETER_PROCESSING` sections. Call of subroutines in the `EQUATION` section requires locally assigned variables, as will be explained in the next section. User defined functions may in turn call other functions, including NMF special functions as `LINEARIZE`, `NMF_ERROR`, and the `EVENT` set of functions that also will be introduced later.

User defined functions can be either local to a single model or global, accessible from all models. The declaration is identical for both types, but local function declarations (there may be several) should be placed just before the `END_MODEL` keyword of the model they belong to.

NMF user defined functions can be divided into four categories:

1. functions in the NMF assignment notation
2. functions in Fortran
3. functions in C
4. functions that are available in the target system

We will present the syntax for function declarations in the Backus-Naur Form (BNF), that is used to describe all NMF syntax. The notation looks strange at first, but once the initial resistance is surpassed, most people tend to often turn to the formal definition of a particular construct. The full NMF syntax description and semantic notes are located in Appendix 1 of the NMF report [Sahlin 1996].

5.2.1 Commented BNF Example: NMF Function Definitions

The following notation applies for the NMF version of Backus-Naur Form:

notation	meaning
<...>	Syntagm

'x..'	Represents literal x..
:=	Define operator
[...]	Optional construct
{...}	Repeat one or more times
{...$...}	Repeat to $ or exit, {a$b} is equivalent to a[{ba}]
(...|...)	Alternative definitions

We start with a sample piece of NMF code, to compare with. Some references (as NMF comments in italics) have been made to relevant syntagms:

```
/*'FUNCTION' <type_spec> <function_name>   '('        <formal_list>                 ')'*/
FUNCTION     VOID  Hom_wall_proc (a, l, lambda, rho, cp, n, c, d, dx)

/*'LANGUAGE' ( 'NMF' | 'F77' | 'C')*/
LANGUAGE     NMF
INPUT
/ *'FLOAT'   <formal_var_list> ';'*/
   FLOAT a, l, lambda, rho, cp;
/*  'INT'   <formal_var_list> ';'*/
   INT n;
OUTPUT
   FLOAT c, d, dx;
CODE
/*   { <body_statement> $ ';' } [';']*/
    dx := l / n ;
    c := rho * cp * dx * dx / lambda;
    d := lambda * a / dx ;
END_CODE
END_FUNCTION
```

NMF MODEL 5-3: FUNCTION Hom_wall__proc

The BNF must be read in a hypertext fashion. Each construction is broken down into smaller parts. These are then broken down further, and so on until elementary pieces remain. We will follow one thread here to illustrate the way: The contents of the formal list of arguments (e.g. `'a, l, lambda, rho, cp, n, c, d, dx'`) is called <formal_list>, which in turn is defined as a comma delimited list of <formal_name>. The list may also be empty (no arguments to the function). <formal_name> is defined to be equal to the syntagm <ident>, which is a sequence starting with a letter, and possibly continuing with <id_char>, which may be a letter, a digit, '_', or '$'.

Text in *italics* are comments to the reader and not part of the formal BNF.

'FUNCTION' <type_spec> <function_name> '(' <formal_list> ')'

```
                'LANGUAGE' <language>
                ['INPUT'        { <formal_decl> } ]
                ['OUTPUT'       { <formal_decl> } ]
                ['LOCAL'        { <formal_decl> } ]
                <body>
                'END_FUNCTION'
<type_spec> :=              ( 'INT' | 'FLOAT' | 'BOOLEAN' | 'VOID' )
```

A function returns a value that must be specified as INT, FLOAT, or BOOLEAN. Subroutines return nothing, and are declared VOID. See Semantic Note no. 13 in Appendix 1 of the NMF Report.

```
<formal_list> :=            [ { <formal_name> $ ',' } ]
<language> :=               ( 'NMF' | 'F77' | 'C')
```

Language specification is required for EXTERNAL functions as well

```
<formal_decl> :=
                (       'INT' <formal_var_list> ';'
                |       'FLOAT' <formal_var_list> ';'
                |       'STRING' <formal_var_list> ';'
                )
<formal_var_list> :=        { <formal_var_spec> $ ', ' }
<formal_var_spec> :=        <formal_name> [ <formal_field_decl> ]
<formal_field_decl> :=      '[' { <formal_field_size> $ ',' } ']'
<formal_field_size> :=      <formal_name>

<body> :=       (       'EXTERNAL'  <external_name> '(' <formal_list> ')'
                |       'CODE'          <code>          'END_CODE'
                )
```

The <body> is either a call to an external function or code in the specified language delimited by CODE and END_CODE. These keywords should be surrounded by carriage returns. (This cannot be deduced from the BNF above).

```
<code> :=       (       { <body_statement> $ ';' } [';']
                |       Code according to the specified non-NMF language
                )
<body_statement> :=  ( <statement> | 'RETURN' <expression> )
<formal_name> :=                <ident >
<ident> :=                  <letter> [ { <id_char> } ]
<id_char> :=                ( <letter> | <digit> | '_' | '$' )
```

Some syntagms are omitted here, i.e. not all threads are complete in the exerpt.

5.2.2 Functions in NMF Notation

The simplest way of writing a small function is generally to use native NMF notation, as in the example in the previous section. Most common is perhaps a body containing just a `RETURN` statement followed by a single expression.

Local variables are not permitted for native NMF functions. This is due to difficulty of allocating dynamic memory for local field variables in standard Fortran 77.

One advantage with native NMF functions in comparison with F77 or C is that are accessible for symbolic processing, i.e. a translator can calculate, e.g., derivatives, whereas this is unlikely to be possible in the near future for functions in other languages.

5.2.3 Functions in F77 or C

NMF may contain source code of functions or subroutines in Fortran 77 or C. A translator will generally not be able to automatically set up a foreign function call, since these often are compiler dependent. It will just emit in a separate file any encountered code that does not correspond to the language that is currently being generated.

The NMF declaration of a foreign function contains information of type and input/output status of each position in the formal parameter list. Variable names for each position can be chosen arbitrarily.

An example of an NMF-wrapped F77 subroutine is:

```
FUNCTION VOID Hom_wall(Qa, Qb, c, d, n, T, T_prime, Out1, Out2)
LANGUAGE F77
INPUT
   INT n;
   FLOAT Qa, Qb, c, d, T[n];
OUTPUT
   FLOAT T_prime[n], Out1, Out2;
CODE
C*************************************************************
C                 F77 SOURCE
C
       SUBROUTINE TQ_HOM_WALL (QA, QB, C_COEFF, D_COEFF, N, T,
     &                         DT_T, TA, TB)
       INTEGER N
       REAL D_COEFF, C_COEFF,
     &      T(N), TA, TB, TAA, TBB, QA, QB, DT_T(N)
* Counters
       INTEGER I
       DO 102 I = 2, (N-1)
          DT_T(I) = (-2.0*T(I)+T(I+1)+T(I-1))/C_COEFF
 102      CONTINUE
       TAA = QA/D_COEFF+T(1)
       DT_T(1) = (TAA+T(2)-2.0*T(1))/C_COEFF
       TA = 0.5*(TAA+T(1))
       TBB = QB/D_COEFF+T(N)
       DT_T(N) = (TBB-2.0*T(N)+T(N-1))/C_COEFF
       TB = 0.5*(TBB+T(N))
       RETURN
       END
C*************************************************************
END_CODE
END_FUNCTION
```

NMF Model 5-4: FUNCTION Hom_wall

5.2.4 Target Environment Functions - EXTERNAL

It is frequently convenient to be able to call any function that is available in the target environment from the NMF code. A typical example of when this is applicable is when a foreign function is available only in binary form. Only an NMF declaration of the actual call is then necessary, for example:

```
FUNCTION VOID Hom_wall2 (Dt, Qa, Qb, c, d, n, Store, Out1, Out2)
LANGUAGE F77
INPUT
   INT n;
   FLOAT Dt, Qa, Qb, c, d, Store;
OUTPUT
   FLOAT Store, Out1, Out2;
EXTERNAL HMWLL (n, c, d, Dt, Qa, Qb, Out1, Out2, Store)
END_FUNCTION
```

NMF Model 5-5: FUNCTION Hom_wall2

Parameters may change order in the external call. See also Semantic Note no. 23 in Appendix 1 of the NMF Report [Sahlin 1996].

Note also that variables in NMF subroutines may act as both input and output. They must then be declared as `A_S` in the calling model (see next section). Functions that return a value may generally not have any `OUTPUT` parameters.

5.2.5 Discussion

Several suggestions have been made to improve the declaration of user defined functions. Some things that most likely will be changed in future versions are:

- Typing of arguments and output in terms of NMF `QUANTITY_TYPES`. This would enable automatic unit checking of NMF code.
- A syntax which is more like that of `CONTINUOUS_MODELS`
- A possibility to have local variables in native NMF functions
- A possibility to have (implicit) equations in native NMF functions

5.3 NMF Variable Assignments

So far, all the models that have been presented have been expressed in terms of equations. Some readers may have examined other NMF models and been puzzled to discover not only equalities '=' but also assignments ':='. Unfortunately, NMF assignments lead to a great deal of misunderstanding and we will try to clarify their background and use.

Pedagogical NMF models should contain an absolute minimum of assignments. The main problem is to make students accustomed to thinking in terms of equations. However, in practical modelling when the fundamentals are clear, NMF assignments occur quite frequently, since they are very useful, and allow us to state additional model properties.

There are two categories of NMF variables that receive their values through explicit assignments in the model rather than by being a part of a global system of equations:

category	NMF role	description
Locally assigned variables	**LOC**	• Locally assigned variables receive their value by assignment. Assignments to LOC variables are in most respects equivalent to equations with a single declared good inverse • Their fundamental job is to carry subroutine output into equation evaluation • They may also occur in some other convenient constructions, where equations are not allowed • They must be assigned to before they are referred • They may remain unassigned, but must not be assigned repeatedly • They may not occur in links • They may not occur as time derivatives
Assigned state variables	**A_S**	• Assigned variables retain their values from one timestep to the next • They are necessary in order to express models that can have multiple modes (or states), such as a thermostat • They must be referred before they are assigned to • They must be assigned to once, never repeatedly • They may not occur in links • They may not occur as time derivatives

5.3.1 Locally Assigned Variables

As we have seen, NMF allows calling of external functions and subroutines. This is an important gateway to already existing models, which can be turned into NMF with a simple "wrapper." It also enables coding of special algorithms that are beyond the scope of straight NMF or that are proprietary in nature and that therefore are distributed in binary form. However, even if the actual model is hidden in a subroutine, (trivial) equations must be declared in the calling NMF model. The basic rules for calling external subroutines are:

1. Subroutine output must be stored in a LOC or A_S variable.
2. Subroutine input may be any expression of present variables and parameters, excluding those that are assigned as output in the same call.

The NMF rule that there must be one explicit equation (declared "=" sign) per `OUT` variable still applies.

Let us pretend for a moment that the finite difference approximation of the wall model from Section 5.1.2 is a trade secret, that we would like to contain in a binary `EXTERNAL` routine. The "wrapper" model could then take the following form:

```
CONTINUOUS_MODEL tq_hom_wall_wrapped

ABSTRACT
   "A 1D finite difference wall model.
    One homogeneous layer.
    TQ interfaces on both sides."

EQUATIONS

      /*Hom_wall was declared in Section 5.2.3 */

 CALL Hom_wall(Qa, Qb, c_coeff, d_coeff, n, State, Temp, Out1, Out2);

   /* wrapper equations */
   FOR i = 1, n
     State'[i] = Temp[i];
   END_FOR ;

   0 = -Ta + Out1;
   0 = -Tb + Out2;

LINKS

/*  type           name      variables ....  */

    TQ             a_side    Ta, POS_IN Qa ;
    TQ             b_side    Tb, POS_IN Qb ;

VARIABLES

/* type     name    role  def  min    max   description*/

   generic  State[n] OUT                    "Internal State vector"
   Temp     Ta       IN   0. abs_zero BIG  "a-side surface temp"
   Temp     Tb       IN   0. abs_zero BIG  "b-side surface temp"
   HeatFlux Qa       OUT  0. -BIG     BIG  "a-side entering heat"
   HeatFlux Qb       OUT  0. -BIG     BIG  "b-side entering heat"
   generic  Temp[n]  LOC                    "Local for State "
   Temp     Out1     LOC                    "Local for Out1  "
   Temp     Out2     LOC                    "Local for Out2  "

MODEL_PARAMETERS

/* type      name    role  def  min  max    description  */
   INT       n       SMP   3    3    BIGINT "no of internal states"

PARAMETERS

/* type      name     role def   min   max  description  */

   Area      a        S_P  10.   SMALL BIG  "wall area"
   Length    thick    S_P  .25   SMALL BIG  "wall total thickness"
   HeatCondL lambda   S_P  0.5   SMALL BIG  "heat transfer coeff"
   Density   rho      S_P  2000. SMALL BIG  "wall density"
   HeatCapM  cp       S_P  900.  SMALL BIG  "wall heat capacity"
```

```
   generic   c_coeff  C_P                    "internal coeff."
   generic   d_coeff  C_P                    "internal coeff."
   generic   e_coeff  C_P                    "internal coeff."

PARAMETER_PROCESSING

/*Hom_wall_proc was declared in Section 5.2.1 */

  CALL Hom_wall_proc(a, thick, lambda, rho, cp, n,
                     c_coeff, d_coeff, e_coeff);

END_MODEL
```

NMF Model 5-6: tq_hom_wall_wrapped

In the wall model, the "wrapper equations" are indeed trivial and therefore seem rather superfluous. Usually, the wrapper contains some meaningful conversion, for example, from one set of units to another.

Note that the `IN/OUT` partitioning of the link variables has been changed in this version of the wall model. The subroutine *input* arguments (Cf. Section 5.2.3) `Qa` and `Qb` have been declared as `OUT` variables of the NMF model to illustrate the fact that subroutine inputs may be any expression of present NMF variables. This will lead to a rather awkward calculation algorithm in an input/output oriented environment, including a call to a numerical equation solver. Depending on other environment characteristics this may or may not influence the overall calculation work in any significant way. Typically, it will have little influence on the overall execution time in a simultaneous solver for real-scale problem sizes.

It is fundamental to understand that subroutine input may depend on model `OUT` variables. However, since this leads to more complicated generated code for some environments, it rarely occurs in practice.

Explicit local assignment of `LOC` variables is equivalent to a subroutine call. The assignment modelling repertoire that was presented in the `PARAMETER_PROCESSING` section (3.7.6) may be used in the `EQUATION` section as well for `LOC` (and `A_S`) variables. A typical structure is that a model is in either on or off mode:

```
/* Excerpt from Annex 17 chiller*/

IF Gamma <= 0  OR  M_fl_c <= 0 OR  M_fl_e <= 0  OR  T_in_e <= T_set
 OR Q_reg <= 0 THEN
          /* The chiller is off */
    T_ex_e := T_in_e ;
    T_evap := 0 ;
    Qevap  := 0 ;
    Power  := 0 ;
    Qcond  := 0 ;
    Cop    := 0 ;
    Status := 0 ;
    Optime := 0 ;
    Power_c  := 0 ;
    H_tran_e := 0 ;
    H_tran_c := 0 ;
```

```
ELSE
      /* The chiller is on  */
 Status :=  IF Q_reg < q_min THEN
                  1
            ELSE_IF Q_reg < q_max THEN
                  2
            ELSE
                  3
            END_IF;

 Qevap  :=  IF Q_reg < q_min THEN
                  q_min
            ELSE_IF Q_reg < q_max THEN
                  Q_reg
            ELSE
                  q_max
            END_IF;

  Optime := IF Q_reg < q_min THEN
                  Q_reg / q_min
            ELSE_IF Q_reg < q_max THEN
                  1
            ELSE
                  1
            END_IF;

  T_ex_e := T_in_e - Qevap / M_fl_e / CP_WAT ;
  Etaev  := 1 - EXP (- auev / M_fl_e / CP_WAT) ;
  T_evap := T_in_e - (T_in_e - T_ex_e) / Etaev ;
  Etacd  := 1 - EXP (- aucd / M_fl_c / CP_WAT) ;
  Ct     := CRTemp (icompr, T_cond, T_evap) ;
  Pl     := Qevap / q_max * Ct ;
  Fl     := PLFACT (icompr, Pl) ;

  T_diff := T_cond - T_evap;
  Cop    := IF  T_diff < -1E-4  OR  T_diff > 1E-4  THEN
                 Fl * etacar * (-ABS_ZERO + T_evap) / T_diff
            ELSE
                 1
            END_IF;

  Power   := Qevap / Cop ;
  Qcond   := Qevap + Power ;
  Power_c := Power * Optime ;
 H_tran_e := Qevap * Optime ;
 H_tran_c := Qcond * Optime ;
END_IF ;
```

Statements enclosed by `IF … THEN … ELSE … END_IF` structures must in NMF be assignments. For equations, only conditional expressions are available.

The previous example, and any other conditional structure, can always be converted into equivalent equations. The only fundamental difference is that in assignment form, a variable may remain unassigned and without a defined value, while in equation form some (dummy) equation must always be in effect.

From Section 3.8 we have

```
IF test <= 1 THEN
    a := 2
  ELSE_IF test > 3 THEN
    a := 4
END_IF;
```

In this case, a (a LOC variable) will not always be defined. The phrase can be converted into an equivalent equation for a (an IN or OUT variable):

```
a = IF test <= 1 THEN
      2
    ELSE_IF test > 3 THEN
      4
    ELSE
      0    /* or some other arbitrary value*/
    END_IF;
```

5.3.1.1 Discussion

The main advantage of the use of local assignments is that the modeller may, in a compact form, give advice about an efficient solution algorithm. Such an algorithm may be nearly impossible to deduce by automatic symbolic methods, since very complex condition structures may have to be successfully analyzed.

For many solvers, the cost of an additional local assignment in a model is nearly negligible, since it will not add to the dimension of the global system matrix. Therefore, any variable that can be turned into a local assignment should. A model is considered to be of better quality if all potential local assignments have been utilized. Examples of variables that could have been converted into locals in previous examples are the virtual temperatures Taa and Tbb of the wall model in Section 5.1.2, and the local pressure drop dp in the leak model of Section 4.4.2.

A limitation of NMF v. 3.02 is that variable limits must be constants. A trick to check a variable against arbitrary limits with the introduction of an additional local variable is the following. Assume that the variable var is to be kept between lo_expr and hi_expr.

The condition can then be written and simplified,

```
lo_expr < var < hi_expr,

0 < var - lo_expr < hi_expr - lo_expr,

0 < (var - lo_expr)/(hi_expr - lo_expr) < 1.
```

Introducing the local variable Test,

```
Test := (var - lo_expr)/(hi_expr - lo_expr)
```

allows us to rewrite the condition with constant limits

```
0 < Test < 1.
```

Current translators assume that assignments have been written in an executable order. It is reasonable to expect that future translators will be more advanced in this respect and be able to check and correct the order of given assignments.

5.3.2 Assigned State Variables

The statement repertoire presented so far is sufficient for systems that lend themselves to an equation description. It is meaningful to extend the set to also include systems that exhibit hysteresis, i.e. that remember previous state in a way that cannot be captured with straight algebraic and ordinary differential equations. The construction chosen in NMF for this gives great freedom to memorize state, also for models that have been externally defined in subroutines.

An assigned state variable remembers its value between timesteps. It is updated by explicit assignment. It must be referred before it is assigned to in a model. For example, let `s` be a variable that has been declared `A_S`. Let the following statement be part of a model that is executed N timesteps and on average K iterations in each step:

```
s := s + 1;
```

The value of `s` at the end of the simulation should be N, i.e. `s` should be reset by the solver (or generated code) to the value of the previous timestep, prior to each iteration.

The most common example for illustrating the use of assigned states is a model of a thermostat:

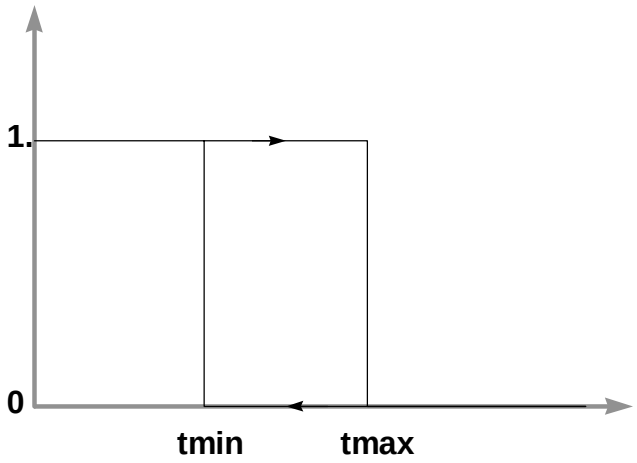

Figure 5-2. Thermostat characteristic

```
    Signal := IF T > tmax  THEN 0
              ELSE_IF T < tmin  THEN 1
              ELSE Signal
              END_IF;
```

We will return to the modelling of systems with multiple modes in the next section. Here, we will present yet another version of the wall model. In the previous version, from Section 5.3.1 , the finite difference approximation was packaged in an external routine, but equations were still integrated by the target simulation environment. Using assigned states, it is possible to store and advance system state entirely within the external routine. The merits of doing this for this particular model is certainly questionable, but there are many other instances where local memory of state and use of a tailored time-advancement are of great value. Examples of such models are dynamic pipe models, where various so called plug-flow models can be very efficient.

The basic rules for such external models are the following:

1. The storage required must be handled by passed NMF assigned state storage space, i.e. it is illegal for an external routine to have its own implementation of storage between timesteps.
2. The size of the current target solver timestep and global time (if necessary) must be passed explicitly to the external routine in the call. All times are in seconds.
3. The external routine must not assume that the size of the timestep is fixed. It may change from very short to very long during the integration.
4. The model may naturally be called multiple times in each timestep. Before each iteration the storage space will automatically be restored to that of the previous timestep by the calling NMF environment.

```
CONTINUOUS_MODEL tq_hom_wall_wrapped2

ABSTRACT
   "A 1D wall model in external routine.
    One homogeneous layer.
    TQ interfaces on both sides."

EQUATIONS

  CALL Hom_wall2(Timestep, Qa, Qb, c_coeff, d_coeff, n, /*input*/
                 Store,                                 /*storage*/
                 Out1, Out2);                           /*output*/

/* wrapper equations */

   0 = -Ta + Out1;
   0 = -Tb + Out2;

LINKS

/*  type           name      variables ....  */

    TQ             a_side    Ta, POS_IN Qa ;
```

```
    TQ              b_side    Tb, POS_IN Qb ;

VARIABLES

/* type     name    role  def  min    max   description*/

   Temp     Ta       IN   0. abs_zero BIG  "a-side surface temp"
   Temp     Tb       IN   0. abs_zero BIG  "b-side surface temp"
   HeatFlux Qa       OUT  0. -BIG     BIG  "a-side entering heat"
   HeatFlux Qb       OUT  0. -BIG     BIG  "b-side entering heat"
   generic  Store[n] A_S                   "Internal storage "
   Temp     Out1     LOC                   "Local for Out1  "
   Temp     Out2     LOC                   "Local for Out2  "

MODEL_PARAMETERS

/* type   name    role  def  min  max    description  */
   INT    n       SMP   3    3    BIGINT "size of internal storage"

PARAMETERS

/* type      name     role def   min   max  description  */

   Area      a        S_P  10.   SMALL BIG  "wall area"
   Length    thick    S_P  .25   SMALL BIG  "wall total thickness"
   HeatCondL lambda   S_P  0.5   SMALL BIG  "heat transfer coeff"
   Density   rho      S_P  2000. SMALL BIG  "wall density"
   HeatCapM  cp       S_P  900.  SMALL BIG  "wall heat capacity"
   generic   c_coeff  C_P                   "internal coeff."
   generic   d_coeff  C_P                   "internal coeff."

PARAMETER_PROCESSING

 CALL Hom_wall2_proc(a, thick, lambda, rho, cp, n, c_coeff, d_coeff);

END_MODEL
```

NMF Model 5-7: tq_hom_wall_wrapped2

5.3.2.1 Discussion

The possibility to package models in external routines with freedom to propagate internal state out of control of the target simulation environment is a debatable issue. The purists feel that the main merit of a symbolic language is that everything should be available for inspection and processing and that external routines defeats this purpose. However, from a more pragmatic standpoint, the freedom bears other advantages:

1. Models may be hidden in binary form for commercial reasons
2. Large existing packages can be accessed with a minimum effort, without re-writing and re-debugging
3. Efficiency can often be gained by using a tailored procedure for particular models

5.4 Multiple Modes and Discrete Events

Many physical systems can be in several distinct modes with different equations governing their behavior in each mode. When certain conditions are met, the system switches from one mode to another. Many models have for example distinct behavior in off, normal, and saturated state.

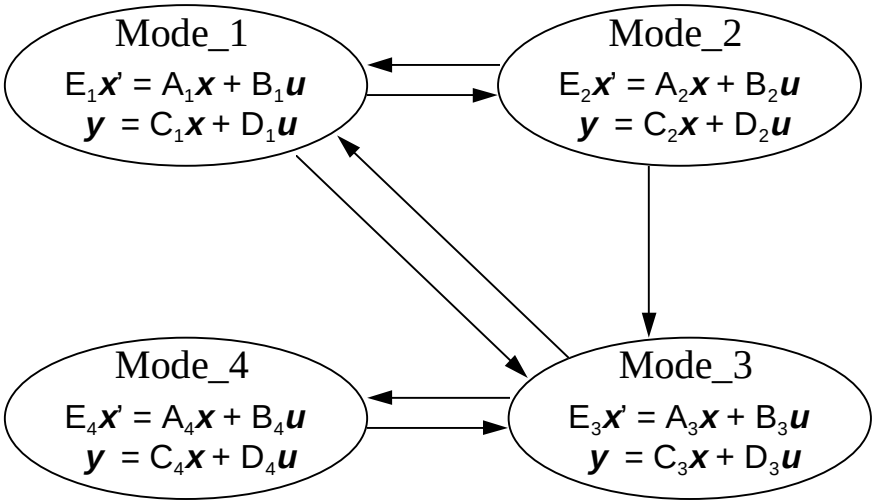

Figure 5-3. A model with four modes, arrows represent possible mode switches. Each switch will have an associated condition.

A fundamental difficulty in the numerical treatment of such models occurs when the model is allowed to switch mode between individual iterations. Then it is very common that the solution is caught in an oscillatory behavior without progress in the direction of the solution.

A trick to avoid this problem is to write model equations in such a way that the state changes are delayed until (global) convergence has been obtained in the previous state. After convergence, switch conditions are evaluated, the switch is done, and the model will be in the new mode in the next timestep. Thus, the switch is delayed until a new timestep is started. To do this, the model equations must be possible to evaluate outside of the mode domain. This is usually a necessary requirement in any model since a solver must be able to rather liberally evaluate equations in various locations as it searches for a solution.

This delay of the switch event can be obtained by using an assigned state. Let us illustrate the technique by completing the thermostat model from the previous section. In an NMF implementation of a thermostat model, the equation that a solver “sees” in each timestep is always smooth; the out signal is kept to a constant:

```
CONTINUOUS_MODEL Thermostat_no_events

ABSTRACT "Thermostat w dead band tmin-tmax,
          OFF=0, ON=1 at low temperature.
          No event calls."

EQUATIONS

    Out_signal = Old_signal; /* Old_signal will be maintained
                                constant in each timestep*/

    /* since Old_signal is an A_S, it should be assigned to
       after being referred to*/

    Old_signal := IF T > tmax THEN
                     0
                  ELSE_IF T < tmin THEN
                     1
                  ELSE
                     Old_signal
                  END_IF ;


LINKS

/* type         name         variables       */

   ControlLink  Out_sign     Out_signal ;
   ControlLink  In_sign      T ;

VARIABLES

/* type  name          role def  min  max  description    */

   generic  Old_signal A_S                "State On or Off"
   generic  T          IN                 "In-signal"
   generic  Out_signal OUT  0    0    1    "Out-signal On=1 or Off=0"

PARAMETERS

/* type    name   role def  min  max  description    */

   temp    tmin   S_P                "lower limit of dead band "
   temp    tmax   S_P                "higher limit of dead band"


END_MODEL
```

NMF Model 5-8: Thermostat_no_events

This model will always overshoot the switch event by some fraction of the current timestep. Thus if nothing else is done, there will be a time step dependent error, which may become large in, e.g., a variable timestep environment. A remedy for this problem is presented in Section 5.4.2.

Hysteresis can occur in unexpected places, and is not always associated with physical friction. Regard for example the `bdleak` model among the Multizone Air Exchange models. A physical leak with finite volume will “remember its state” by the properties (temperature) of the air in the leak cavity.

5.4.1 A General Framework for Multimode Models

NMF does not allow separate equations for each mode. A single set of equations must be valid at all times, and conditions must be contained in these. Thus multimode equations should be written in the following general form:

```
0 = IF Mode == 1 THEN
    <expression 1>
    ELSE_IF Mode == 2 THEN
    <expression 2>
    ELSE_IF Mode == 3 THEN
    <expression 3>
    ELSE
    <expression 4>
    END_IF
```

The update of the assigned state `Mode` is done after the actual equations have been declared. The set of switches in Figure 5-3 can, e.g., be implemented in the following way:

```
IF <1->2  expression>  >=  0 AND Mode == 1 THEN Mode :=2 END_IF;
IF <1->3  expression>  >=  0 AND Mode == 1 THEN Mode :=3 END_IF;
IF <2->3  expression>  >=  0 AND Mode == 2 THEN Mode :=3 END_IF;
IF <2->1  expression>  >=  0 AND Mode == 2 THEN Mode :=1 END_IF;
IF <3->1  expression>  >=  0 AND Mode == 3 THEN Mode :=1 END_IF;
IF <3->4  expression>  >=  0 AND Mode == 3 THEN Mode :=4 END_IF;
IF <4->3  expression>  >=  0 AND Mode == 4 THEN Mode :=3 END_IF;
```

where the *expressions* are assumed to be formulated in such a way that zero is crossed from negative to positive when the switch shall occur. This construction allows insertion of arbitrary algorithmic code at switches. We will motivate the chosen construction further in the next section, where we turn our attention to the signaling of discrete events to a solver.

5.4.2 Signaling Discrete Events

The method presented for multimode modelling may lead to an overshoot of the event, since the switch is delayed until the end of the current timestep. This is a problem with a variable timestep solver, which, if everything else is behaving smoothly, may take very long steps.

For this reason, many solvers are able to detect event signals from the model. When such a signal has been detected, the solver can go into an event-location mode, and with more or less sophisticated methods, determine the precise location of the switch and finish the timestep just before. After a short step across the discontinuity, normal time stepping is resumed.

Three NMF functions are dedicated to event signaling, `EVENT`, `EVENTN`, and `EVENTP`. They take two arguments, an assigned state variable and a signal (an expression), the zero crossing of which signifies an event. `EVENTP` flags an event only when the signal

goes from negative to positive, EVENTN from positive to negative, and EVENT in any direction. The event functions will compare the present value of the signal with the (in the A_S) memorized past value, signal the event if necessary, update the A_S variable, and return the current signal value.

It is important to make sure that the event functions are actually updated as necessary, and not lie buried in some logical structure. Hence the proposed event switching construction in the general framework, where all event functions are updated in every iteration. Tailored constructions are obviously applicable in individual cases.

```
IF Eventp(G[1], <1->2 expr>) >= 0 AND Mode == 1 THEN Mode :=2 END_IF;
IF Eventp(G[2], <1->3 expr>) >= 0 AND Mode == 1 THEN Mode :=3 END_IF;
IF Eventp(G[3], <2->3 expr>) >= 0 AND Mode == 2 THEN Mode :=3 END_IF;
IF Eventp(G[4], <2->1 expr>) >= 0 AND Mode == 2 THEN Mode :=1 END_IF;
IF Eventp(G[5], <3->1 expr>) >= 0 AND Mode == 3 THEN Mode :=1 END_IF;
IF Eventp(G[6], <3->4 expr>) >= 0 AND Mode == 3 THEN Mode :=4 END_IF;
IF Eventp(G[7], <4->3 expr>) >= 0 AND Mode == 4 THEN Mode :=3 END_IF;
```

Both Mode and G must be declared as A_S variables.

5.4.2.1 Discussion

The requirement of an explicitly declared memory variable in the event functions may be perceived as unnecessary. Surely this variable could be introduced automatically by a translator. This is true, and "auto-expandable" versions of the event functions could easily be furnished with future NMF versions. However, the main motivation for the present construction is that event calls may also occur in external code. The same is true for the chosen method of modelling hysteresis, and for any other NMF system routine, as LINEARIZE or NMF_ERROR. NMF's ability to accommodate sophisticated external models is an important feature which is often lost when typical models in different formalisms are compared and discussed.

The final version of the thermostat model with event calls can be examined in Appendix 2 of the NMF report. Notice the explicit calls to event functions to ensure that they are always evaluated. The event functions are special in that they have an I/O argument (the A_S). In general, functions are allowed to return only a single value.

6. Solved NMF Problems

6.1 Exercise 1 - a Basic Mechanics Problem

Formulate an NMF-library for basic mechanics in one dimension, containing a point mass, a spring and a damper. Set up an input file for some target environment for the following problem.

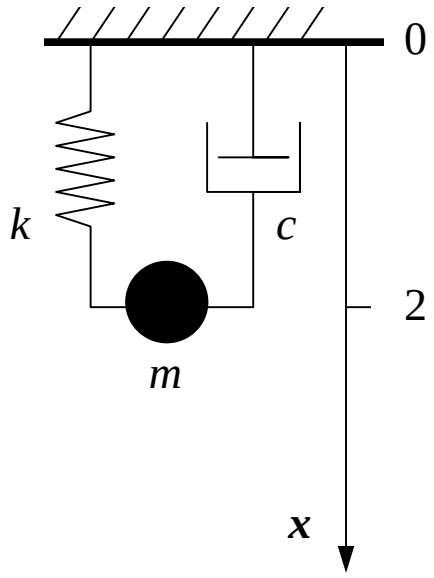

Figure 6-1. Simple mechanical system

6.2 Solution to Exercise 1

6.2.1 NMF Models

A point mass is characterized by its mass, a spring by its original length and its spring constant and a damper by its damping coefficient. These quantities will be parameters in the models. The variables of the problem will be the position and velocity of the point mass, the position of both ends of the spring and damper and the forces acting on the point mass and on the end of the spring and the damper.

We start by defining quantity types for these quantities. Standard SI units seems to be the best choice for units. Now to the question about which quantity types are `CROSS` and which are `THRU`. This information is not used for parameters and the choice for these is therefore mainly of future interest (in the event they are used as variables). Here they should all be defined as `CROSS`, since none of them is of an obvious flux type. The variables are less obvious. Both velocity and force have a direction with respect to some global coordinate system associated with them, which may seem to indicate `THRU` type. However, the question one should ask is whether it would be appropriate to apply Kirchoff's law to any of these variables. The answer is no, it would not. Hence all variables are defined as `CROSS`.

```
QUANTITY_TYPES

/* name            unit          kind */
   Length          "m"          CROSS
   Velocity        "m/s"        CROSS
   ElastModule     "N/m"        CROSS
   DampCoeff       "Ns/m"       CROSS
   Mass            "kg"         CROSS
   Force           "N"          CROSS
```

We also need a link type to be able to link the different components. It should contain both position and the force.

```
LINK_TYPES

/* name               quantity types... */
   FL               (Force, Length)
```

The dynamics of the point mass is governed by Newton´s law. This is a second order ordinary differential equation. Since second order derivatives is not part of the NMF syntax we have to define the velocity of the point mass.

```
CONTINUOUS_MODEL point_mass

ABSTRACT "Point mass"

EQUATIONS

   0 = X' - V;          /* definition of velocity */

   0 = m * V' - SUM i=1,n F[i] END_SUM; /* Newton's law */
```

It would be nice to be able to allow any number of forces to act on the point mass. Therefore, we need a vector interface. Note that the order of the variables of the link has to be the same as in the definition of the link type (force, length).

```
LINKS

/* type          name          variables... */
FOR i=1,n
   FL          terminal[i]     F[i], X
END_FOR;
```

Now it is time to define the variables and parameters of the point mass. We have to have the same number of `OUT` variables as the number of equations. If the models are to be used without alteration in an input-output oriented environment, we must at this

stage formulate some principles about which variables that should be given (IN) and which should be calculated by the model (OUT).

Any IN/OUT selection is acceptable in this case for NMF. However, if we chose to give X as input, we will get a so called “high index” problem, which is difficult to solve in most environments unless the equations are differentiated symbolically. A discussion about these issues can be found in [Brenan 1989].

Let us pick the forces as IN, to avoid the high index problem. This leaves position and velocity as OUT for the mass. For an input-output oriented environment, this means that any component that is connected to the mass must have force as OUT and position as IN.

```
VARIABLES

/* type     name role [def [min   max]]  description */
   Length   X    OUT   0    -BIG  BIG    "position"
   Velocity V    OUT   0    -BIG  BIG    "velocity"
   Force    F[n] IN    0    -BIG  BIG    "force on terminal i"

MODEL_PARAMETERS

/* type     name role [def [min   max]]  description */
   INT      n    SMP   1    0     BIGINT "number of terminals"

PARAMETERS

/* type     name role [def [min   max]]  description */
   Mass     m    S_P   1    SMALL BIG    "mass"

END_MODEL
```

NMF Model 6-1: point_mass

The models for the spring and the damper are rather straightforward, and it does indeed seem possible to have force as OUT and position as IN.

```
CONTINUOUS_MODEL spring

ABSTRACT "An ideal linear spring"

EQUATIONS

   L = X2 - X1;          /* definition of length */

   F1 = k * (L - l0);    /* Hooke's law */

   F2 + F1 = 0;          /* Force balance */

LINKS

/* type    name    variables... */
 FL    terminal_1   F1, X1;
 FL    terminal_2   F2, X2;

VARIABLES
```

```
/* type  name  role  [def  [min  max]]  description */
 Length   L     OUT   1    -BIG  BIG    "length of spring"
 Length   X1    IN    1    -BIG  BIG    "position of terminal 1"
 Length   X2    IN    2    -BIG  BIG    "position of terminal 2"
 Force    F1    OUT   0    -BIG  BIG    "force on terminal 1"
 Force    F2    OUT   0    -BIG  BIG    "force on terminal 2"

PARAMETERS

/* type      name   role  [def [min  max]]  description */
 Length       l0    S_P   1   0   BIG   "length of spring"
 ElastModule  k     S_P   1   0   BIG   "elasticity module"

END_MODEL
```

NMF Model 6-2: spring

```
CONTINUOUS_MODEL damper

ABSTRACT "An ideal linear damper"

EQUATIONS

 L = X2 - X1;    /* definition of length */

 F1 = c * L';    /* damping equation */

 F2 + F1 = 0;    /* Force balance */

LINKS

/* type  name  variables... */
 FL  terminal_1  F1, X1;
 FL  terminal_2  F2, X2;

VARIABLES

/* type  name  role [def [min max]] description */
 Length  L     OUT    1  -BIG BIG  "length of damper"
 Length  X1    IN     1  -BIG BIG  "position of terminal 1"
 Length  X2    IN     2  -BIG BIG  "position of terminal 2"
 Force   F1    OUT    0  -BIG BIG  "force on terminal 1"
 Force   F2    OUT    0  -BIG BIG  "force on terminal 2"

PARAMETERS

/* type   name  role [def [min max]] description */

 DampCoeff c    S_P    1  0  BIG  "damping coefficient"

END_MODEL
```

NMF Model 6-3: damper

The models above have been written without locally assigned (`LOC`) variables. Let us contemplate for a moment whether any variable could be converted to `LOC`. No variable that occurs on a link is a candidate and this excludes most variables in the present example. We are left with `V` of the mass, and the lengths `L` of the spring and the damper. `V` and the length of the damper occur time differentiated and this excludes

them as well. We are left with the length of the spring, which could indeed be converted into a local. To do this, the following changes would have to be made. First, the equation for L must be changed to an assignment:

```
  L := X2 - X1;          /* definition of length */
```

Then the declaration of the variable must be changed to reflect the new role.

```
VARIABLES

/* type  name  role  [def  [min  max]]  description */
 Length   L     LOC   1   -BIG  BIG   "length of spring"
```

6.2.2 Input file for IDA Solver

Below is an input file for IDA Solver for the system configuration of Exercise 1. The `IN`/`OUT` partitioning of the variables in the NMF models have, as previously discussed, been selected to avoid `IN-IN` and `OUT-OUT` connections for the given configuration. However, in an input-output free environment like IDA Solver any connection is permitted, and springs and dampers can for example be connected in series.

Some comments have been added to clarify meanings of file statements.

```
ABSTRACT
"Simple test of exercise 1
mechancis library"

!logical and physical files
FILES
  OUTPUT OUTPUT1
    PATH MECH.PRN
END_FILES

! declare model instances
! P - point mass
! S - spring
! D - damper
MODULES
  MODULE P
    TYPE Point_mass
! parameters for P
n 2
m 1

  MODULE S
    TYPE Spring
k 1
l0 1

  MODULE D
    TYPE Damper
c 1
END_MODULES

!Connect the instances. Variable
!level connections are used in
!this example
CONNECTIONS
  P.F(1) = S.F2
  P.F(2) = D.F2
  P.X = S.X2
  P.X = D.X2
END_CONNECTIONS

!Define boundary values
BOUNDARIES
S.X1 0
D.X1 0
END_BOUNDARIES

!Define initial values
START_VALUES
DEFAULT 0
P.X 2
P.V 0
D.X2 2
D.L 2
S.X2 2
S.L 2
END_START_VALUES

!Specify integration data
INTEGRATION
  FROM 0.
  TO 10.
!Initial time step to be used
  STEP 0.0001
!Overall error tolerance
  TOL 0.001
!Switch limit between relative
!and absolute tolerance
  TOL_LIM 1.

!Specify output data
  LIST
    OUT_ALL
    OUT_TIMES
      0 1 1
    END_TIMES

    P.X Position
    S.F1 Spring_force
    D.F1 Damper_force
  END

  LOG
    FILE OUTPUT1
      P.X Position
      S.F1 Spring_force
      D.F1 Damper_force
  END
END_INTEGRATION
```

7. References

Andersson, M., 1990, *An Object-Oriented Language for Model Representation*, Licentiate Thesis, Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden, May 1990, CODEN:LUTFD2/(TFRT-3208)/1-102/(1990)

Bonneau, D., F.X. Rongere, D. Covalet, B. Gautier, 1993, *CLIM 2000 - Modular Software for Energy Simulation in Buildings*, Conference proc. Building Simulation'93, IBPSA, Adelaide, Australia, Aug. 1993

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

Bring, A., P. Sahlin, 1993, *Modelling Air Flows and Buildings with NMF and IDA*, Conference proc. Building Simulation'93, IBPSA, Adelaide, Australia, Aug. 1993

Bring, A., L. Eriksson, H. Hermansson, M. Lindgren, P. Sahlin, 1995, *IDA - an Environment for Building and Energy Systems Simulation*, Conference proc. Building Simulation'95, IBPSA, Madison, WI, USA, Aug. 1995

Buhl, W.F., E. Erdem, F.C. Winkelmann, E.F. Sowell, 1993, *Recent Improvements in SPARK: Strong Component Decomposition, Multivalued Objects, and Graphical Interface*, Conference proc. Building Simulation'93, IBPSA, Adelaide, Australia, Aug. 1993

Clark, D.R., 1985, *HVACSIM+ Building Systems and Equipment Simulation Program - Reference Manual*, National Institute of Standards and Technology, Washington D.C.

Clarke, J.A., D.F. Mac Randall, 1993, *The Energy Kernel System: Form and Content*, Conference proc. Building Simulation'93, IBPSA, Adelaide, Australia, Aug. 1993

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

Elmquist, H., 1978, *A Structured Model Language for Large Continuous Systems*, Phd thesis, Lund Institute of Technology

Elmqvist, H., **1986,** *LICS: Language for Implementation of Control Systems*, Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden

Grozman, P., P. Sahlin, 1996, *ASHRAE RP-839 NMF Translator - User's Guide*, ASHRAE Inc. and Bris Data AB

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

Jeandel, A., F. Favret, E. Lariviere, 1993, *ALLAN.Simulation - a General Software Tool for Model Description and Simulation, Conference proc*. Building Simulation'93, IBPSA, Adelaide, Australia, Aug. 1993

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., 1987, *Reflections about Representation Methods, proc. workshop on the future of building energy modelling*, Ispra, Italy, Nov. 1987, CEC EUR 11603 EN PREPRINT, May 1988

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 NMF to SPARK*, Conference proc. Building Simulation'95, IBPSA, Madison, WI, USA, Aug. 1995

Ohlsson, B., A. Persson, 1991, *Sandys - a simulation program for electrical circuits and systems*, ABB Review 5/91

Pelletret, R., 1994a, Personal communication

Pelletret, R., S. Soubra, 1994b, *Standardizing Model Documentation - The PROFORMA Experience*, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R., S. Soubra, W. Kielholz, 1995, *Transferring Simulation Techniques to End-Users - Application to TRNSYS*, Conference proc. Building Simulation'95, IBPSA, Madison, WI, USA, Aug. 1995

Sahlin, P., A.Bring, 1993, *Applying IDA to Airflow Problems in Buildings*, ITM report 1993:3

Sahlin, P., A. Bring, 1995, *The IDA Multizone Air Exchange Application*, Bris Data AB and Div. of Building Services Engineering, KTH

Sahlin P., A. Bring, K. Kolsaker, 1995, *Future Trends of the Neutral Model Format*, Conference proc. Building Simulation'95, IBPSA, Madison, WI, USA, Aug. 1995

Sahlin, P., A.Bring, E.F.Sowell, 1996, *The Neutral Model Format for Building Simulation*, Version 3.02, Report, Dept. of Building Sciences, KTH, Stockholm (available at ftp://urd.ce.kth.se/pub/reports/nmfre302.rtf)

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, E.F. Sowell, 1989, *A Neutral Format for Building Simulation Models*, proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

Shapovalov, A., 1995, personal communication

Yi Jiang , Zhenxi Xu, Feng Chen, 1994, *TUHVAC - an object oriented computer software for HVAC design, analysis and simulation*, preprints of the 4th Intnl. Conf. on System Simulation in Buildings, Dec, 1994, Liege, Belgium

8. APPENDIX A A System Model Example in Future NMF (v. 4)

An example of a system model in NMF version 4. The syntax below has been formally approved as an example of NMF v. 4 by the NMF Committee.

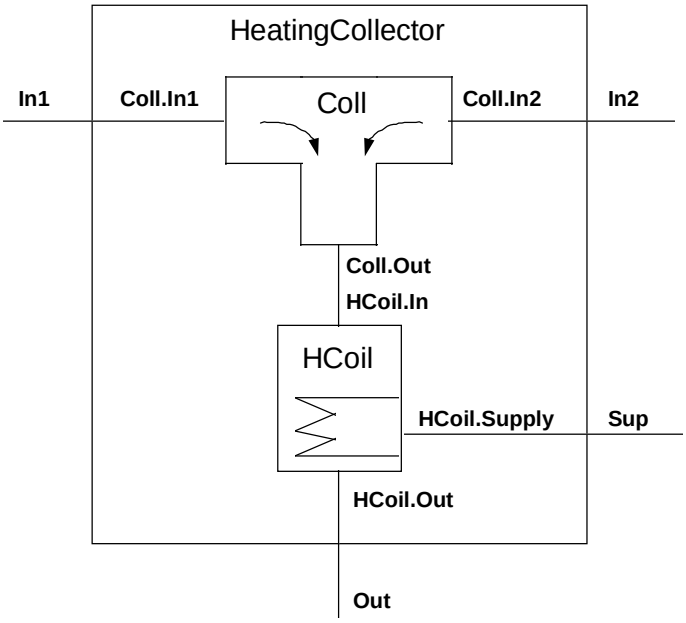

Figure. A-1. System model with a collector and heating coil

```
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;

DOCUMENTATION
// No documentation yet
END_DOCUMENTATION

END_MODEL
```