HVAC COMPONENT MODEL LIBRARIES FOR EQUATION-BASED SOLVERS

Edward F. Sowell
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92634

Michael A. Moshier
Department of Mathematics and Computer Science
Chapman University
Orange, CA 92667

ABSTRACT

This paper discusses development of a library of equation-based models for building HVAC system simulation. The main source of the models is the ASHRAE Secondary Systems Toolkit (Brandemuehl 1993), augmented with primary system models from other sources in order to provide a library sufficient to model conunon HVAC systems. While the target of this work was the Simulation Problem Analysis Research Kernel (SPARK), a number of issues arising in construction of the library are of interest whenever models are to be expressed in equationbased, non-algorithmic form Among other results, it is shown that models of this nature benefit from finer modular granularity than equivalent component-oriented models, and that there are important issues of connectability and computational efficiency that can be addressed with a new approach to modeling units of physical measure.

INTRODUCTION

Component-oriented simulation environments long have been used to construct computer models of building heating and cooling systems (Klein, Beckman, and Duffie 1976, Park, Clark, and Kelly 1985). The principal attraction of this approach has been modeling flexibility, whereby an analyst could assemble a model configured to match the actual building system rather than being constrained to selection from a fixed set of preprogrammed system models. Recent developments in non-algorithmic (i.e., equation-based) modeling (Klein 1991, Sahlin and Bring 1991), equation-based solving (Buhl et al. 1993), component model standardization (Sahlin and Sowell 1989), and model structuring principles (Mattsson 1988) now promise to further the flexibility and efficiency of the component-oriented approach. In addition to stimulating development of entirely new component-oriented modeling tools, it can be anticipated that many of these emerging ideas will find their way into existing component oriented simulation environments, as well as new versions of the whole-building simulators.

While many users of the component-oriented environments routinely construct their own component models for special elements, a basic library of standard elements, such as cooling coils and fans, has been of fundamental importance in their wide-spread acceptance. Likewise, a comprehensive library adapted to the newer technology will be required. Such libraries will differ from existing libraries in several fundamental ways. First, the models will be non-algorithmic so that the greater flexibility of input/output free modeling (Mattsson 1988) can be allowed in model formulation. Second, the models will be expressed at a finer level of granularity so as to maximize code reuse and maintainability. Beyond this, these models eventually will be expressed in a standardized, neutral format (Sahlin and Sowell I989) so they can be automatically translated to the expected variety of simulation environments. Achieving these goals raises a number of interesting issues, including optimal model granularity, submodeling, units of measure, validation, code maintenance, and solution efficiency.

LIBRARY CONTENT

Library content, in terms of kinds of components represented and the level of model detail, depends upon intended use. Because the development of SPARK and related work traditionally has been focused on building energy conservation, the basic SPARK HVAC library is aimed at energy analysis over an annual period using a one-hour time step. Consequently, the equipment models of interest are steady-state mass, energy, and moisture balances. Pressure-flow relationships are not included; instead, flow rates are to be viewed as determined by control laws or specified inputs. Also, the library model for zone loads is a simple air heat balance, relying on time-varying load profiles from other environments such as DOE-2.

The basic library is intended to support modeling of HVAC systems typically employed in commercial buildings such as constant volume reheat, variable air volume reheat, and dual duct systems. These systems deliver a heated and/or cooled mixture of return and outside air to the conditioned spaces. Heating of the air streams is accomplished with hot water coils (heat exchangers) or electric resistance heaters, while cooling is by means of chilled water or direct expansion coils which also provide dehumidification. Fans and damper boxes set air flow rates, either fixed or modulated according to a control strategy. The set of components dictated by this requirement, Table 1, is a subset of the ASHRAE Secondary Systems Toolkit, augmented by chiller, boiler, and cooling tower models from DOE-2.

Table 1. SPARK HVAC Library

Air System
Fan
Mixing box
Diverter
Heating coil
Cooling coil
Air-to-air Heat Exchanger
Direct Evaporative Cooler
Indirect Evaporative Cooler
Humidifier
Zone
Water System
Pump
Valve
Controls
Economizer
Proportional
Heat/Mass transfer
NTU Heat Exchanger
Dry Coil
Wet Coil
Bypass factor
DX Unit
Psychrometrics
Saturation Temperature
Enthalpy
Humidity Ratio
Dew Point
Relative Humidity
Specific volume
Moist Density
Wet Bulb
Primary equipment
Boiler
Chiller
Cooling Tower
Systems & Integration
Constant Volume Reheat
Variable Volume Reheat
Dual Duct

LIBRARY STRUCTURING PRINCIPLES

Beyond content, a library is characterized by the principles upon which it is structured. For example, the ASHRAE Toolkit is based on the principles of algorithmic models and modularity, as expressible in FORTRAN. Also, it represents relatively course granularity. Other characteristics include use of SI units, and use of mass flow, temperature, and humidity ratio as the primary interfacing variables. The structuring principles used in the SPARK library and supporting rational are developed below.

Interface Variables and Units

Customarily, a library is based on a common set of interface variables and units of physical measure. This is necessary, because otherwise the component models could not be easily interconnected. Interface sets in common use within the HVAC modeling community include (Air volume flow rate, Temperature, Relative Humidity), (Mass flow rate of dry air, Temperature, Humidity Ratio), and (Mass flow rate of dry air, Enthalpy, Humidity Ratio). The first set is of greatest familiarity to U.S. HVAC practitioners, but is not well suited to modeling because these are not the quantities appearing in the conservation equations, necessitating conversions during the simulation. The last set is closest to the conservation laws, but enthalpy is less familiar to some practitioners than temperature. The second set is a compromise. The mass and humidity variables can be used directly in the conservation equations. Although enthalpy must be calculated for these conservation equations, the temperature is directly available for the heat transfer relationships. It has the added advantage of being the set employed in the ASHRAE HVAC 2 Toolkit. Units of the interface variables is a separate question. If it were expected that U.S. HVAC practitioners would be the main users of SPARK, the English IP system would be preferred. However, most researchers in the field would probably prefer SI units. All considered, it was decided to use the second set as component interface variables in SI units for the SPARK basic library. This is not ideal because of varying user needs, and conversion objects, both for psychrometrics and units of measure, must be used. We shall return to these questions in New Directions below, where we propose a better alternative for future development.

Model Expression

It is widely recognized that any simulation environment for building energy systems requires models for the same basic set of physical components as given in Table 1. Ideally, there should be a single representation made accessible to all simulation environments. The Neutral Model Format (NMF) (Bring, Sahlin, and Sowell 1992) fills this need, providing a rigorous, non-algorithmic syntax for mathematical model definition that can be automatically translated to various environments. Indeed, prototype translators have already been implemented for SPARK and IDA (Kolsaker 1993).

For the above reasons it was initially proposed that the NMF be used to express the basic component model library in this project. Unfortunately, it became necessary to temporarily set this decision aside and express the library directly in SPARK syntax. The reasons for this decision were twofold. First, it became apparent that the translator would not be ready in time to support model development; experience has shown that models must be tested, and as a practical matter testing cannot be carried out on code that cannot be executed. The second reason is that there is great advantage to extensive use of submodeling in equation-based models, and as of this writing the NMF does not support a submodeling facility. It is hoped that these two shortcomings of the NMF can be resolved and that this library can later be reexpressed in the NMF.

Equation-based Modeling

Today, most computer models are expressed as a sequence of a assignments. Such models are called *algorithmic* or *input/output oriented* because they relate a prescribed set of outputs to a prescribed set of inputs by means of a prescribed algorithm. Such models are now recognized as being limited in flexibility when compared to equation-based models. The distinction is that although the physical laws being enforced are the same, with algorithmic modeling every selected input set requires a separate algorithmic model. In principle, there may be as many as $n_v!/(n_v-n_e)!\ n_e!$ different algorithmic models for a single mathematical model involving n_v variables and n_e equations, differing only in the specified input set.[1] Clearly it is better to express the model one time, strictly in terms of the underlying equations. In this form, the model itself is free from any designation of inputs and outputs. This is called an *equation-based model* because the constraints among variables are expressed by equality rather than assignment. The term *non-algorithmic* is also appropriate because no sequence of evaluation or assignment to variables is implied, both of which characterize algorithmic models. At time of use of the model, one particular input set is specified, thus defining a problem. Assuming that the problem is well-posed, equation oriented solving environments (e.g., SPARK and IDA) generate an algorithm for solution.

Submodeling and Model Granularity

Equation based models can be packaged in several different ways, and one important distinguishing feature is submodeling capability. Another is the fineness of model granularity when submodeling is employed. For example, in SPARK (Buhl et al. 1993, Buhl, Sowell, and Nataf 1989) each individual equation is a separate object, resulting in fine granularity. This we call the *equation-object* approach. However, the equation objects are most often encapsulated in *macro objects,* which can refer to both equation objects and other macro objects. Thus the macro object affords a submodeling capability for SPARK. Importantly, however, the macro objects are disassembled to equation objects prior to simulation, so the fine granularity is exposed to the solver.

An alternate approach is to package all equations related to a particular physical component in a single module while still retaining the non-algorithmic property. This approach, which we might call *equation-modular*, is employed in IDA. A third approach, sometimes called *simultaneous-modular* (Chen and Stadtherr 1985), also packages multiple equations together, but the modules are algorithmic and the solver sees only the interface variables. The latter method is not truly equation based, but the flexibility of arbitrary input/output set designation is attained by forming residual equations based on assigned module output variables. Although not yet described in the literature, the most recent version of TRNSYS is reputed to work on this basis. With either the equation-modular or the simultaneous modular approach the equations themselves are not individually accessible to the solver, hence they are characterized as having coarse granularity.

The finer granularity of the equation-object approach offers solution efficiency advantages over the coarser granularity approach. This is because any equations and variables hidden within modules are not available to the solving environment. In contrast, the equation-object approach exposes all equations and variables, thus allowing automatic construction of efficient solution algorithms. This is demonstrated in the SPARK methodology. Briefly described, this method employs matching on a bipartite graph to match equations to variables, followed by discovery of a small set of iteration variables (a cut set) in the directed graph representing data flow in the problem. With this approach, the solution process can iterate on fewer variables, offering potentially dramatic efficiency gains.

The fineness of granularity of the equation-object approach also enhances maintainability and code reuse. For example, an HVAC library must implement a wide variety of psychrometric functions. Among these functions are several key relationships upon which all others depend, such as

[1] Some of these may not represent well-posed problems.

the equation of state, the relationship between vapor pressure and temperature for water at saturation, and the enthalpy equation. However, there are several mathematical models for representing these basic functions, so one might ask how difficult it is to switch to a different model. When the equation object approach is employed a decision to change from, say, the ideal gas law to the Virial equation of state is implemented by the substitution of a single new object. Moreover, there is maximum code reuse since the equation code exists in a single place. In contrast, conventional practice often repeats code for each equation where needed.

It should be noted that some of the advantages of fineness inherent in the equation-object approach can be attained in the equation-modular and simultaneous-modular approaches by using functions. For example, the ubiquitous psychrometric relations can be expressed as functions, thus making them available for multiple use and centralizing their maintenance. However, functions are normally expressed algorithmically so that whenever a relationship is needed with alternate input/output sets it is necessary to have multiple implementations. For example, if we sometimes need *Psat(Tsat)* and other times need *Tsat(Psat)* two functions are required in spite of there being a single underlying mathematical model.

EXAMPLES

The application of the above structuring principles can be demonstrated by looking at subsets of the total library. Here we show a partial implementation of the psychrometric and heat exchanger subsets.

Psychrometrics

Psychrometrics are fundamental to HVAC models. The ASHRAE Handbook of Fundamentals (ASHRAE 1993) gives the basic relationships. An example of the equation-object implementation is provided by the dew point relationships:

P_w = psat(t) (1)
w = MW_RATIO * $P_w / (P_{atm} - P_w)$ (2)

Equation (1) is the relationship between water vapor pressure P_w and dry bulb temperature *db* at saturation, while Eq. (2) is the relationship among humidity ratio *w*, vapor pressure, and atmospheric pressure P_{atm}. Each is implemented as an equation object in the SPARK library, as shown below:

```
/* SPARK object satpress.obj */
define satpress(t, p)
      double t[T], p[P];

      p = pwsdb(t);
      t = dbpws(p);
```

```
/* SPARK object humratio.obj */
define humratio(patm, pw, w)
      double patm[P], pw[P], w[w];

            w  = wfpwp(pw, patm);
            pw = pwfwp(w, patm);
```

Note that the argument list defines the interface to the object as well as their units. In the body of the object each interface for which there is a convenient explicit inverse is equated to a function representing this inverse. For example, *wfpwp(pw, patm)* is Eq. (1), while *pwfwp(w, patm)* is the same equation, but solved for *pw*. The functions can either be expressed as C functions, or can be interpreted at run time. The inverses can be generated symbolically by a computer algebra tool such as MACSYMA. In fact, the SPARK MACSYMA interface automatically generates the SPARK object file and the C functions for the inverses, given only the object equation in the form <expression> = <expression> (Nataf and Winkelmann 1992).

The dew point object can then be implemented as a *macro object*, which just connects equation objects together for modeling convenience. In the SPARK syntax this is:

```
/* SPARK object dewpt.obj */
macro
declare  humratio  hr;
declare  satpress  sp;
      link  PwDB(hr.pw, sp.p)
      link  patm(hr.patm)
      link  db(sp.t)
      link   w(hr.w)
```

Here we see *humratio* and *satpress* objects instantiated with **declare** statements. The first **link** statement equates the *p* interface of the *satpress* object with the *pw* interface of the *humratio* object. This will be recognized as the essence of the dew point concept. The common vapor pressure is named *PwDB* by the **link** statement. The other **link** statements each have a single interface in their list. These therefore do not form connections like the first **link**, but still serve to "name" the variables within the macro object, effectively elevating them to be interfaces to the macro object. Thus this macro object can be connected into other macros objects (or SPARK problems) through *patm*, *db*, *w*, and *PwDB*, although the last would not be viewed as a proper interface since it plays only an internal role.

Another interesting psychrometric relationship is that for wet bulb temperature. This property for any psychrometric point, say *(db, w)*, is defined as the dry bulb temperature at the intersection with the saturation curve of a line of constant enthalpy passing through *(db,w)*. Using the superscript * to represent this intersection point, the following simultaneous equations define the wet bulb temperature property:

```
h = CP_AIR * db + w * (CP_VAP * db
+ HF_VAP)                         (3)
hs* = CP_AIR*t* +
ws**(CP_VAP*t*+HF_VAP)            (4)
ws* =  MW_RATIO * pws*/(patm -
pws*)                             (5)
pws* = psat(t*)                   (6)
h + (ws* - w) * CP_WAT * t*= hs*
                                   7
```

Equations (3) and (4) define the enthalpy at *(db, w)* and *(t^*, ws^*)* respectively. Equations (5) and (6) ensure that *(t^*, ws^*)* lies on the saturation curve. Finally, Eq. (7) is an energy balance of air undergoing an isenthalpic process from *(db, w)* to *(t^*, ws^*)* (ASHRAE 1993).

It is seen that Eqs. (3-6) define important psychrometric properties, namely enthalpy, humidity ratio, and saturation pressure (or temperature). As such, they find wide use other than defining wet bulb temperature and are therefore natural candidates for equation objects in the library. Equation (7), however, defines no new property and is unlikely to be useful other than here. Nonetheless, we define it as an equation object in the library because SPARK currently has no construct for equations outside equation objects. For want of a better name, we call it *eq31* after its numbering in the ASHRAE Handbook.

Assuming then that we have equation objects for each of the above equations, the wet bulb macro object is expressed as:

```
/* SPARK object wetbulb.obj */
macro
declare  enthalpy e1, e2;
declare  humratio  hr;
declare  satpress  sp;
declare  eq31  eq31;
        link  patm(hr.patm)[P]
        link  db(e1.db)[T]
        link  h(e1.h, eq31.h)[h]
        link  w(e1.w,eq31.w)[w]
        link  hs_star(eq31.hs_star,
              e2.h)[h]
        link  ws_star(eq31.ws_star,
              e2.w, hr.w)[w]
        link  t_star(eq31.t_star,
              e2.db, sp.t)[T]
        link  pws_star(hr.pw,
              sp.p)[P]
```

Figure 1 shows a diagrammatic representation of the wet bulb macro object. Some SPARK users benefit from such a diagram during development of macro objects. Others are comfortable with a simple textual representation of the equations in the mathematical model, i.e., Eqs.(3-7).

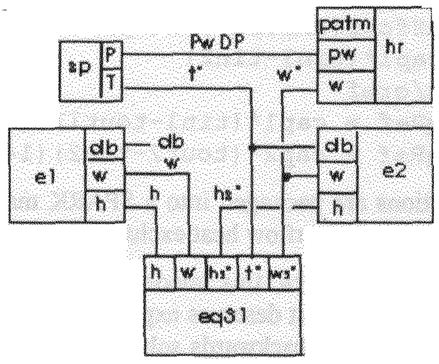

Figure 1. Diagrammatic representation of the wet bulb object.

It is instructive here to consider other advantages of equation-object modeling. In conventional modeling practice, Eqs. (3-7) are algebraically reduced to a single equation which is then solved iteratively when the model is used. There are several disadvantages of this approach. For one thing , the underlying model is obscured. More importantly, the benefits of code reuse are not realized. For example, if one later wished to use a more accurate enthalpy model, the wet bulb model would have to be changed. Or worse yet, its hidden use might cause the wet bulb object *not* to be changed, resulting in differing definitions of enthalpy within the model. With the fine granularity of the equation-object approach, a change of a single enthalpy object in the library would immediately take effect in wet bulb as well as all other models in which it plays a role.[2] Finally, it will be observed that the iterative solution process is not part of the wet bulb model. Instead, when wet bulb occurs in a problem, the needed iteration can be handled by the global solution process.

Heat Exchangers

Heat and mass exchange devices play key roles in HVAC systems. Heating and cooling coils are liquid to air heat exchangers, heat recovery is done with air to air heat exchangers, and in central plant equipment there are numerous examples of liquid to liquid heat exchangers. All of these devices can be modeled with the same general model of a heat exchanger.

The basis of all heat transfer equipment models in the library (as in the ASHRAE Toolkit) is the well

[2]We concede that customary algorithmic, modular modeling could offer the same advantage provided that the importance of fine granularity was observed with abundant use of functions.

known Ntu-Effectiveness model (Kays and London 1984). Expressed in equation-object form we have:

```
ntup = ua/cap1                   (8)
cRatiop = cap1/cap2              (9)
effp = effctr(cRatiop, ntup)    (10)
qRef = cap1*(tin1-tin2)         (11)
effp = q/qref                   (12)
effp * qRef = cap1*(tin1-tout1) (13)
effp * qRef = cap2*(tout2-tin2) (14)
```

These equations are packaged into a SPARK macro called *htxctr*, a counterflow heat exchanger.

The difference between this model and the one presented in the Toolkit deserves explanation. First, in order to facilitate "backwards solving" usage, as required in design calculations, the Kays and London model was revised to use the first flow stream *Cap1* as a reference instead of *Cmin*. This eliminates the *minimum* function which is non-invertable. Additionally, instead of a single effectiveness function with a configuration parameter to select the flow arrangement, we implement five separate effectiveness objects, giving five different heat exchanger models differing in the declaration of the class for the effectiveness object. Justification for this is partially on grounds of improving invertability. As a result, we are often able to solve for heat exchanger size from design point specifications without iteration. Additionally, the implementation avoids the need for a configuration parameter which would have to be passed up through interfaces of any submodels of which the heat exchanger becomes a part. Actually, the heat exchanger becomes a submodel of several other component models. These include the air-to-air heat exchanger, heating coil, dry cooling coil, and (modified for enthalpy) in the wet cooling coil. All of these are modeled as macro objects.

NEW DIRECTIONS

In the course of this project we observed a number of opportunities to improve model structuring, possibly leading to better future model libraries. Here we shall discuss three that appear to be particularly important, namely the handling of dimensions, units of measure, and fluid properties.

Polymorphic Dimensional Analysis

As seen above, SPARK supports a simple *type checking* system that permits the model writer to enforce compatibility of units. For example, in our library of components, a heat exchanger can only be connected via links that have the appropriate units: mass flow, temperature, and so on. At the same time, SPARK also supports a form of *polymorphism*, whereby a link declared to have *generic* units suppresses the type checking. This allows general use of a single class, e.g., *product*, or *sum*, where otherwise a separate object class would be required for each set of units.

However, dimensional analysis is a powerful check on the physical realizability within a system of equations, and this check is lost with the generic unit concept. There is nothing to prevent linking two lengths and a volume to the three generic links at the interface of a *product* object. Thus we are encouraged to either abandon the present generic link unit, or to enrich the type checking to produce a stronger form of polymorphism called *dimensional polymorphism*. We propose to enrich the type checking system, in the three following ways: **1.** Replace the notion of generic link with the notion of a "dimension variable." This would allow an object to have "generic links" that still require some matching of types; **2.** Give the type checking system access to derived dimensions. For example, the links to the *product* object might be declared to have the dimensions Q, R, and Q*R. That would yield one of the fundamental constraints of dimensional analysis; **3.** Incorporate the algebraic laws pertaining to dimension in the type checking system.

Without going into details, the *sum* object could then model the heat balance equation as (with unit terms in brackets):

```
q1[Q] = q0[Q] + dq[d(Q)]
```

Here we show the use of the dimensional difference operator *d()* which derives a new unit based on the difference between quantities of some given unit. This is also used to distinguish between temperature units and difference in temperature units, as in the *conductance* object

```
q[Q] = u12[Q/d(T)]*(t1[T] - t2[T])
```

Any attempt to connect this object to other links would require that the various terms be *unifiable;* there must be a solution to a system of Abelian group equations that associates with such connections. Coupling Abelian group unification with a mechanism for giving names to derived units, e.g., *area* = *length*^{2}, would give SPARK a tool for constructing and using generic models in a way that guarantees the level of physical realizability that one expects of dimensional analysis.

Fluid links and unit conversions

A related problem is posed by *units,* as distinct from *dimensions*. By the word unit, we understand a specific metric associated with a dimension, such as meters or kilograms. A similarly related problem is how to handle the *complexes* of quantities that represent such things as moist air.

A familiar problem in modeling is that of standard units. Should we express quantities in SI or English units? This has a trivial solution in most cases: simply pick one and enforce it throughout the model. The problem becomes more complicated if we wish to facilitate the integration of models that may have already been constructed on a different standard. The user must then know that when linking an English object to an SI object a conversion object must be interposed. This is a nuisance to the user and adds to execution time and problem size. The impact could be relieved by choosing a standard, say SI, and then wrapping any English object inside the needed conversions. In other words, we could produce new macro objects that simply couple English objects with the needed conversions, thus presenting SI units at the interfaces. The problem with this, however, is what happens to the run time problem when two thus modified English objects are connected. The run time model would include equations to convert from English to SI and immediately back to English, causing two unnecessary equations.

A similar, and perhaps more serious, situation obtains between models involving psychrometric quantities. To see this, recall that dry bulb temperature, humidity and enthalpy are related, giving two degrees of freedom. Now, certain processes are more easily modeled in terms of enthalpy and humidity (e.g., mixing boxes), whereas others are more easily modeled in terms of temperature and humidity (such as heat transfer). Here the exact same problem arises for the psychrometric quantities as it does for mismatched ordinary units. Model writers are faced with the same poor choices: force the user to know internal details (whether a model uses enthalpy or temperature internally) that have no bearing on the physics of the component; or standardize and incur hidden, spurious equations when connecting components actually needing no conversion.

These two situations pose exactly the same mathematical problem, i.e., they represent projections of coordinates from one subspace to another of similar dimension. We will not review here the elements of projective geometry, but simply point out that if we solve the problem of mismatched standards for psychrometric quantities, then we will solve the problem of mismatched ordinary units. One possibility that shows promise is to allow *complexes* of links to be bundled under a single name, and then associated with a system of *unification* equations. For example, an air complex link might include links for temperature, humidity and enthalpy, along with the needed system of psychrometric equations, while a temperature link might include three simple links for Kelvin, Celsius and Fahrenheit, along with the needed conversion equations.[3] To generalize, the equations associated with a complex link may be called conversion equations.

While these ideas could be applied to other modeling environments as well, they are particularly well suited to SPARK. One implementation would be to construct the problem graph in the normal way, then "prune" it to remove objects that produce variables not used elsewhere or reported. This is an easy task with well known graph traversal algorithms.

These ideas for complex links with on-demand conversion equations is still in its early stage, but we are confident that it can be implemented. Conversion equations with polymorphic dimensional analysis will allow the model developer to express the physical content of a model free of the mundane concerns about standard interfaces and unit conversions.

CONCLUSIONS

We have reported on an effort to convert an existing modular, algorithmic HVAC component library to an equivalent equation-object representation, targeted primarily for the SPARK environment. We have attempted to explain the choices available during the course of this development, and to rationalize the decisions taken. Arguments were presented for the advantages of the fine-grain, equation-object approach, as opposed to the more coarse-grained equation-modular approach to equation-based modeling. While both offer the advantages of input/output free modeling, the former additionally offers opportunities for improved "back solving" without iteration, thus reducing run time, and better code reuse and maintainability. In addition, the project called to attention several issues of model structuring, such as handling of units and dimensions in model expressions. An approach based on unification and projective geometry was outlined. We believe this approach solves not only problems with unification of ordinary dimensions and units, but also connecting diverse component model implementations in which interface variables include complexes of variables, such as moist air properties, in an ideal manner.

ACKNOWLEDGEMENTS

This work was sponsored by the Assistant Secretary for Conservation and Renewable Energy, Office of Building Technologies, Building Systems and Materials Division of the U.S. Department of Energy

[3]This is similar to the idea of property links recently suggested by Kolsaker for the NMF.

under Contract No. DE-AC03-76SF00098. Lawrence Berkeley National Laboratory. We would also like to thank the organizers and participants of the Building Simulation '95 Conference for the opportunity to share our work.

REFERENCES

ASHRAE. *Handbook of Fundamentals.* Atlanta: Am. Soc. of Heating, Refrigerating, and Air-conditioning Engineers. 1993

Brandemuehl, Michael J. *HVAC 2 Toolkit: A Toolkit for Secondary HVAC System Energy Calculations.* Joint Center for Energy Management, University of Colorado. ASHRAE 629-RP. 1993.

Bring, Axel, Per Sahlin, and Edward F. Sowell. *The Neutral Model Format for Building Simulation.* Royal Institute of Technology, Dept. of Building Services Engineering, Stockholm. Bul. 24. 1992.

Buhl, W. F., A. E. Erdem, F. C. Winkelmann, and E. F. Sowell. "Recent Improvements in SPARK: Strong Component Decomposition, Multivalued Objects, and Graphical Interface," In *Proceedings of Building Simulation '93* (Adelaide) International Building Performance Simulation Association. Available from Soc. for Computer Simulation International, San Diego, CA, 283-90. 1993.

Buhl, W. F., E. F. Sowell, and J-M. Nataf. "Object-oriented Programming, Equation-based Submodels, and System Reduction in SPANK," In *Proceedings of Building Simulation '89* (Vancouver, BC) International Building Performance Simulation Association, 141-146. 1989.

Chen, H-S., and M. A. Stadtherr. "A Simultaneous-Modular Approach to Process Flowsheeting and Optimization." *AIChE Journal* 31 (11) : 1843-1855. 1985.

Kays, W. M., and A. L. London *Compact Heat Exchangers.* 3rd ed. New York: McGraw-Hill Book Co. . 1984.

Klein, S. *Engineering Equation Solver (EES).* F-Chart Software. 1991.

Klein, S. A., W. A. Beckman, and J. A. Duffie. "TRNSYS- A Transient Simulation Program." *ASHRAE Transactions* 82 (1) : 623-33. 1976.

Kolsaker, K. "Recent Progress in Fire Simulation using NMF and Automatic Translation to IDA," In *Building Simulation '93* (Adelaide, Australia) International Building Performance Simulation Association, 555-560. 1993.

Mattsson, S. E. "On Model Structuring Concepts," In *Proceedings of the 4th IFAC Symposium on Computer-aided Design in Control Systems (CADCS)* (Lund) 1988.

Nataf, J-M., and F. C. Winkelmann. *Automatic Code Generation in SPARK: Applications of Computer Algebra and Compiler-compilers.* Simulation Research Group, Lawrence Berkeley Laboratory. NTIS, LBL-32815. 1992.

Park, C., D. R. Clark, and G. E. Kelly. "An Overview of HVACSIM+, a Dynamic Building/HVAC Control Systems Simulation Program," In *Proceedings of the First Building Energy Simulation Conference, Aug. 21-22.* (Seattle, WA) International Building Performance Simulation Association. 1985.

Sahlin, P., and A. Bring. "IDA Solver- A Tool for Building and Energy System Simulation," In *Proceedings of Building Simulation '91* (Nice, France) Society for Computer Simulation, International, San Diego, CA, 339-348. 1991.

Sahlin, P., and E. F. Sowell. "A Neutral Format for Building Simulation Models," In *Proceedings of Building Simulation '89* (Vancouver, BC) International Building Performance Simulation Association, 147-54. 1989.