Proc. IBPSA, *Building Simulation 2001*,
Rio de Janeiro, September 2001

LBNL-47975

AUTOMATIC UNIT AND PROPERTY CONVERSION IN SPARK

Edward F. Sowell
California State University, Fullerton
Fullerton 92834 USA

Michael A. Moshier
Chapman University
Orange, CA 92866 USA

Ender Erdem
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

May 2001

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy Office of Building Technology, State and Community Programs, Office of Building Research and Standards. of the U.S. Dept. of Energy, under Contract No. DE-AC03-76SF00098.

AUTOMATIC UNIT AND PROPERTY CONVERSION IN *SPARK*

Edward F. Sowell
California State University, Fullerton
Fullerton 92834

Michael A. Moshier
Chapman University
Orange, CA 92866

Ender Erdem
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

ABSTRACT

In object based models, conversion becomes necessary when there is a mismatch among the representations of same physical quantity at the ports of different objects that need to be connected. The simplest example of this is with regard to physical units of measure, such as temperature. A more complex situation, but with the same character, is with regard to fluid flow, where the state can be represented in terms of any pair of independent properties, and the flow rate can be expressed volumetrically or in terms of mass. In HVAC applications, the most common example is moist air where properties and flow rate can be represented by temperature, relative humidity and volumetric flow, or by enthalpy, humidity ratio and mass flow rate, or other combinations. The need to connect objects with such disparate interfaces arises whenever system model developers following different conventions attempt to share models.

Customarily, conversion is done in an *ad hoc* manner, introducing conversion objects here and there in the problem, as needed. A somewhat more structured approach is to rigidly enforce a common set of units among all classes within a particular application-area library; this approach is recommended for the Neutral Model Format (NMF) (Bring, Sahlin et al. 1992). With this approach one can either undertake “hard conversion” of the underlying equations and re-implement the model, or take the easier path of “wrapping” the offending objects in new “macro objects” with needed converters. Neither choice is very attractive. The first is labor intensive, and errors may be introduced into sound code, while the second can lead to many unnecessary conversion, complicating the model and compromising run-time efficiency.

In this paper we show a unified approach to automatic interface matching that does not suffer the above disadvantages. Special automatic conversion links, or “autolinks,” are introduced for this purpose. An autolink carries all admissible representations of one or more variables plus a set of conversion objects that represent constraints that need to be enforced among them. Herein we show that this approach is easily implemented, makes use of existing models with diverse units or properties at the interface, and produces optimum run-time efficiency.

The idea of autolinks has been mentioned before in the literature. Kolsaker and Sahlin make essentially the same suggestion (Sahlin, Bring et al. 1995), but limit their discussion to “property links” only. Sowell and Moshier independently develop the concept, discussing it in terms of unit matching as well as fluid flow and properties. They also introduce the idea of automatic elimination of unneeded conversion objects before run-time by “pruning” the computation graph (Sowell and Moshier 1995). In the following, we expand upon the previous work, using examples to clarify the approach, and show the first implementation in the context of the current features of the Simulation Problem Analysis and Research Kernel (SPARK). We then suggest future extensions to made problem specification with autolinks more intuitive.

OBJECT BASED MODELING

Here we describe models in terms of interacting objects. It is assumed that these objects, and the system models formed from them, have no presumed direction of information flow. This is sometimes called non-algorithmic or input/output free representation. The latter designation is significant in that it means that the model itself is independent of specified inputs or outputs. It is not until a particular input set is specified that a problem is defined and information flow direction is established. The same model can therefore support a wide class of different problems without reprogramming. This style of model representation has been advocated in the modeling literature for some time because it properly places emphasis on the underlying mathematical model rather than upon a

particular solution algorithm (Sowell and Low 1982; Sowell, Taghavi et al. 1984; Konopasek and Jayaraman 1985; Sowell, Buhl et al. 1986; Mattsson 1988; Sahlin 1988; Sahlin and Sowell 1989).
As a simple example, consider the equation

$$C = 5(F - 32)/9 \tag{1}$$

This can be represented graphically with an object as shown in Figure 1.

Here *tc* labels a temperature conversion object. Typically, such an object would be created as an instance of a template or *class*, which embeds Equation 1. *F* and *C* are interface variables, or *ports*, of the object. T_F and T_C are *problem variables*, which are linked to the ports. Port labels relate to the connected problem variable names in the same way that parameters relate to actual arguments in conventional programming. That is, at run time it is as if T_F and T_C were substituted for *F* and *C* in Equation 1.

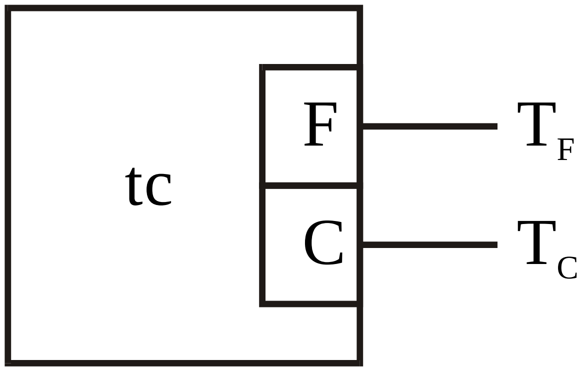


Figure 1: Input/output free model object.

This single object comprises a simple system model. If we then designated T_F as the input, a problem would thereby be defined. Solving the problem would entail solving Equation 1 for the corresponding T_C . Conversely, one could specify T_C as input, forming a problem in which T_F is to be solved for. There would be no change to the *tc* object.

Problems with more than one object are more common. Figure 2 shows a model for system with a fan and a heater. In the graphic for each object, the physical inlet and outlet are shown at the left and right respectively. In the physical system, the fan outlet is connected to the heater inlet. Consequently, we link the fan exit enthalpy *h*, humidity ratio *w*, and mass flow rate *m* to the corresponding ports of the heater inlet. However, this does not imply directionality of information flow in the model solution process; such directionality is not established until an input set is specified.

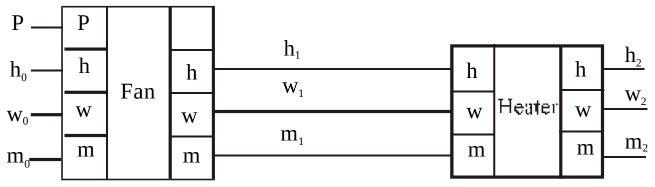


Figure 1: Linking objects

Problem variables are sometimes called *links* because they appear as labels on links among object ports.

MACRO OBJECTS AND MACRO LINKS

Modeling convenience is enhanced when model objects closely correspond to physical components. For example, if a component in the physical system is a heater with an in-built fan, we may prefer a *heaterFan* model object rather than a separate model of each. On the other hand, fan and heater model objects are also sometimes needed separately. The concept of *macro objects* allows us to have both. A macro object is an object that is constructed from other objects like a system model, but has external ports and otherwise looks like and behaves like a simple "atomic" object. This is sometimes called *hierarchical modeling,* and is supported by SPARK and other recent modeling environments. Figure 3 demonstrates this idea, creating a *heaterFan* macro object from the fan and heater objects.

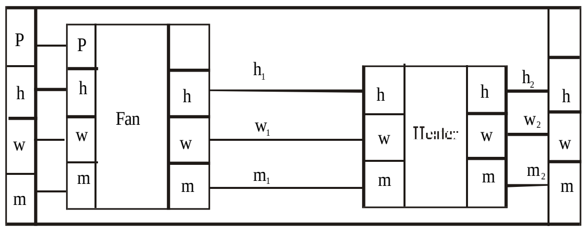


Figure 2: Macro object.

MACRO LINKS

In many physical system models the objects interact through fluid flow, so that flow and properties of a fluid at the exit of one object apply at the entrance of another. When this is the case, model diagrams tend to have linkage groups such as mass flow rate, humidity ratio, temperature, and pressure connecting object after object, making model construction tedious and error prone. Figure 2 is a typical example in an HVAC context, where the properties are enthalpy and humidity ratio, and flow rate is represented as mass per unit time. The concept of a *macro link* is helpful in such models. With this idea, the links for the separate but related variables, e.g., flow and fluid properties, are treated as a single modeling entity. This is depicted in Figure 4 for the same system represented in Figure 2, where the macro link contains sublinks for enthalpy, humidity ration, and mass flow rate.

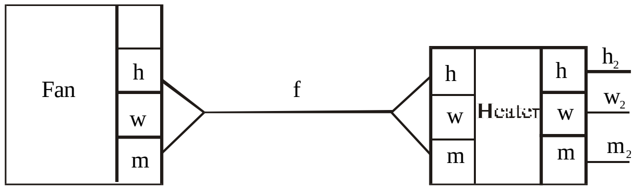


Figure 3: Macro links.

The macro link concept has appeared in the HVAC modeling context several times over the years. Sowell et al. defined networks of HVAC component models in which all linkage was in terms of a single 5-tuple containing mass flow rate, enthalpy, humidity ratio, fresh air fraction, and pressure (Sowell, Silverman et al. 1981). More recently, Sahlin, Bring and Sowell generalized and formalized the idea with the definition of LINKS as an NMF modeling entity (Sahlin and Sowell 1989; Bring, Sahlin et al. 1992). The current version of at least two equation-based simulation environments, SPARK and IDA incorporate the idea in some form (Sahlin and Bring 1991; Sowell 1998).

It should be noted that macro links require a supporting construct at object interfaces, called *macro ports*. Macro ports, i.e., a grouping of interface variables, fills this need. However, since the one implies the other, it is not necessary to have syntax for both macro link and macro port in a language to support the concept. For example, if the fan and heater objects in Figure 4 are defined with *(h, w, m)* macro ports, there is no ambiguity if *f* is merely designated as a *link* rather that a *macro link*. SPARK takes advantage of this, omitting macro link as a syntactic entity in favor of macro ports.

AUTOLINK CONCEPT

We now turn our attention to the problem of maintaining consistent units of measure for links and ports in models of the nature discussed above. The fundamental ideas will be presented in terms of a simple thermal model of a two layer wall, Figure 5. On the left, Layer A, we have a mineral fiber insulation, while on the right there is a layer of concrete aggregate. We will assume that we have thermal models for each, representing the conductivity as a nonlinear function of the average temperature within the layer.

As a baseline case for later comparison, first suppose that both layers are implemented in terms of SI units and are modeled by the equations:

$$\begin{aligned} \overline{T} &= (T_1 + T_2)/2 \\ T_d &= T_1 - T_2 \\ k(\overline{T}) &= k_0 + aT_d + bT_d^{\,2} + cT_d^3 \\ q &= k(\overline{T})T_d \,/\, l \end{aligned} \tag{2}$$

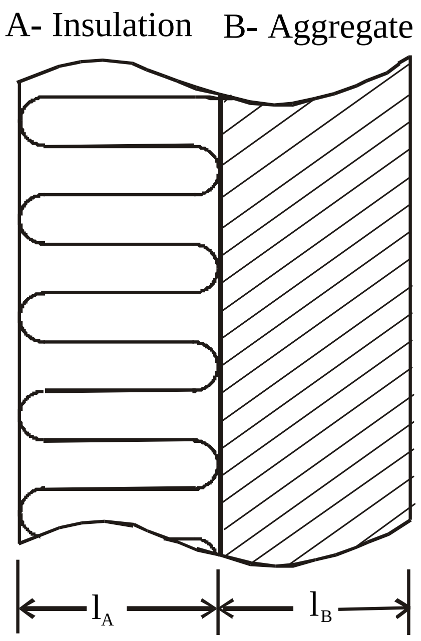


Figure 4: Heat transmission through 2-layer wall

where

$\overline{T}$	= Average temperature [C]
T_d	= Temperature difference [C]
$k(T)$	= Temperature dependent thermal conductivity [W/m-C]
a, b, c	= Correlation coefficients for the material, based on measurements in SI units.
q	= Heat flux [W/m^2-C degree]
T_1, T_2	= Left and right face temperatures [C]
l	= Layer thickness [m].

A model of the system can be represented as two interconnected objects, *LayerA* and *LayerB*, as shown in Fig. 6. Each of these objects embeds an instance of Equations 2, specialized for the particular material of the layer. Note that $\overline{T}$, T_d , and k, as well as their defining formulas, are internal to the object, so at the level of expression shown in Figure 2 there is a single constraint among the interface variables q, T_1, T_2, and l. Due to the assumed temperature dependency of the material conductivity, the constraint is nonlinear. The coefficients a, b, and c are determined by the material and may be different for each layer. These coefficients are also built into the objects, meaning that each object is an instance of a different class representing a particular material.[1]

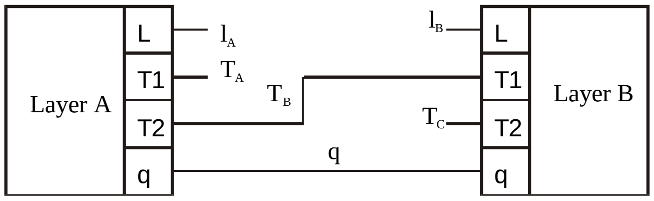


Figure 5: Two layer wall model.

This model can represent several different problems depending upon which variables are specified as inputs. For this example, we choose to specify the two outer surface temperatures, T_A and T_B, the thickness of layer B, l_B, and the common heat flux q. Thus the problem is to determine the interface temperature T_B and the thickness of the insulation, l_A. A later section discusses how solution is carried out in SPARK (See spark implementations).

We shall now modify the preceding example to create a unit mismatch between the models for the two layers. Assume now that Layer B, the concrete aggregate material, is modeled in IP rather than SI units.

That is,

$\overline{T}$	= Average temperature [F]
T_d	= Temperature difference [F]
$k(T)$	= Temperature dependent thermal conductivity [Btu/h-ft-F]
a, b, c	= Correlation coefficients for the material, based on measurements in IP units.
q	= Heat flux [Btu/h-ft^2-F]
T_1, T_2	= Left and right face temperatures [F]
l	= Layer thickness [ft].

[1] We acknowledge that this is not necessarily the best modeling practice. It would not be difficult to implement a single class to represent all layers. We choose this form to exemplify the unit conversion process.

One approach to the difficulty this presents is to introduce four unit converters, one for each interface variable, as shown in Figure 7. In a problem as simple as this one, this is probably the best approach. It is easy to do, leaves the original models intact, and suffers only minor computational overhead. However, if there are many places where this mismatch occurs, manual insertion and connection of the converters becomes tedious and error prone. To circumvent this, one might consider defining a new object class that "wraps" the IP object and the converters, as indicated by the dashed lines in Figure 7.

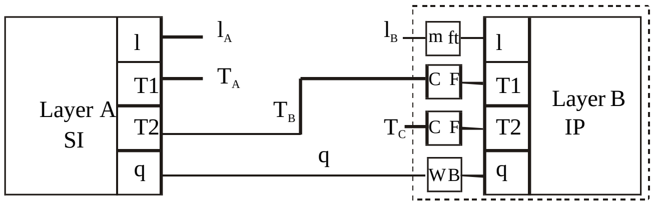
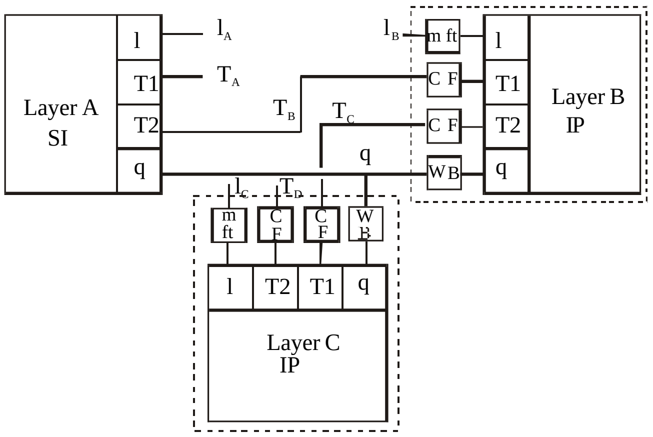


Figure 6 Two layer wall with unit mismatch.

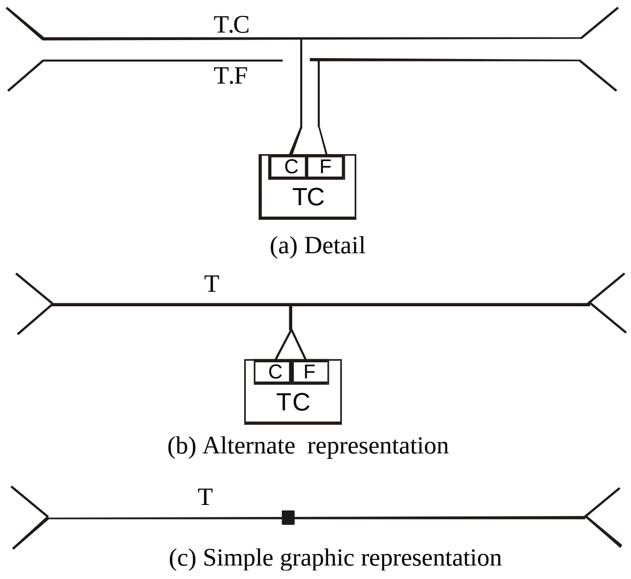
The new class would offer SI ports and could be instantiated throughout the system model as needed. But the problem with this idea is that that there may a third layer, also originally expressed in IP but with the SI wrapper. We would then have the situation shown in Figure 8.



The obvious disadvantage here is that unnecessary conversions are being done. The common temperature between layers B and C, T_C, is converted from F to C and back to F, even though the C units are not used at all, and q is converted from W/m^2 to Btu/h-ft^2 twice. While this may not be an intolerable inefficiency, it can be avoided without modeling inconvenience by using *autolinks*.

Figure 7 Three layer wall with mixed units.

In concept, an autolink is simply a group of related links along with constraint equations among the variables represented on the links. For example, a temperature autolink has two constituent links, Celsius (C) and Fahrenheit (F), and an object representing the relationship in Equation 1. Figure 9 shows the temperature autolink.


T.C
T.F
C
F
TC
(a) Detail
T
C
F
TC
(b) Alternate representation
T
(c) Simple graphic representation

Part (a) of the figure shows the internal details, while (b) and (c) show alternative, simplified representations for use in model diagrams. In the detailed view, the upper line represents the temperature in Celsius, while the lower one represents the same temperature in Fahrenheit. The object labeled *TC* contains the Celsius to Fahrenheit conversion equation, i.e., a constraint between F and C.

Figure 8 Temperature autolink.

To see how the autolink concept is superior to the conversion strategy described in the previous section, let us consider only the TC linkage in Figure 8, connecting the T1 port of C to the T2 port of B; both are in Fahrenheit. Figure 10 shows this using the detailed representation of an autolink from Figure 5.

Since both ports are in Fahrenheit, the F part of the TC autolink, TC.F is connected to both. This leaves the C port of the TC temperature converter unconnected.

Consequently, it can be “pruned” from the computation graph before run-time, leaving this part of the problem without conversion. (See pruning the computation graph below.)

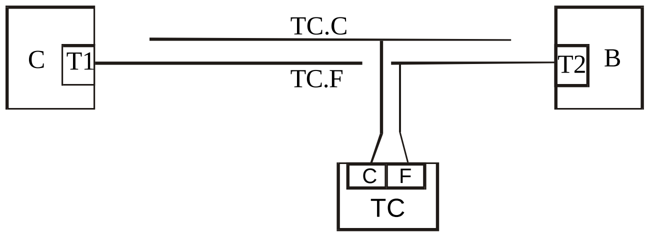


Figure 9: Using autolink for TC.

Now, let us consider the *q* link in Figure 8, where it was observed that two conversions were being done when one should do. Figure 10 shows only the *q* ports from Figure 8, but connected with the autolink idea.

Here we see that a *q* autolink connecting the *q* ports of *A*, *B* and *C* introduces a single heat flux converter, *qc*. Since *A* is in SI units, its *q* port is connected to the W (Watt) part of the *q* autolink, *q.w*, while the B (Btu/h) part connects to the IP ports of the two IP-based objects, *C* and *qc*. Thus we have again reduced the number conversions in the problem.

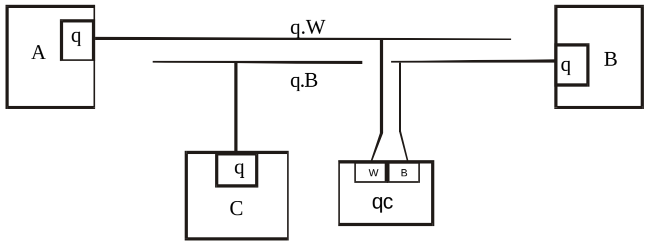


Figure 10: Using autolink for q.

The preceding examples demonstrate the value of autolinks in reduction of model complexity and run time efficiency. However, the true utility of the concept depends somewhat on how convenient it is for the model developer to use. It develops that the macro objects and link ideas make it possible to implement autolinks conveniently.

In Figure 12 we show the two layer wall model from Figure 6 using macro objects and macro links. Here we have embedded the A (SI) and B (IP) objects in macro objects A’ and B’, each with appropriate SI-IP macro ports. As can be seen, the thickness, temperature, and heat flux macro ports have subports *(m, ft)*, *(C,F)* and *(W,B)* respectively. In each, the SI subport is listed first, and the equivalent IP subport second. Since A has an SI interface, there are links to the SI subports in the A’ macro object. Conversely, there are links to the IP subports in the B’ macro object.

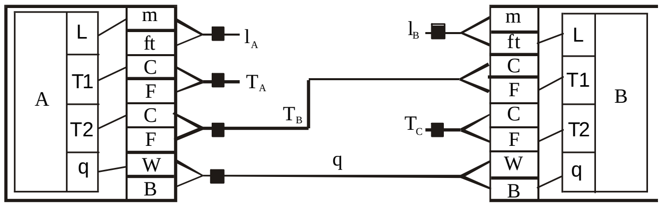


Figure 11: Two layer wallmodel with autolinks and macro ports.

Externally, the macro ports of the two macro objects are connected with autolinks. We use the simple graphic representation for the autolinks, but as can be seen in Figure 9 each is really nothing more than a macro link with an incorporated conversion object.

Thus we see that if the modeling language supports macro objects and macro links, it also supports the autolink concept. In a later section we show syntax for expressing autolinks in SPARK. But first, we show that exactly the same concept serves as fluid property links.

FLUID PROPERTY AUTOLINKS

A fluid property autolink, as described by Sahlin et al. (Sahlin, Bring et al. 1995) is a modeling entity the links ports of fluid flow components carrying fluid flow rate and all relevant properties, plus necessary constraint equations among the properties. Constraint equations derive from fluid physics and are necessary whenever the link is specified to carry more fluid properties than necessary to uniquely describe the thermodynamic state of the fluid.

The macro link in Figure 4 is a simple example of a fluid link, but one in which no constraint equations are necessary, because enthalpy and humidity ratio uniquely determine the state of moist air. However, if a third, redundant property is added to the link, e.g., dry bulb temperature, the psychrometric constraint equation $h = f(T, w)$ must be added. The need to add redundant properties arises whenever the model employs fluid flow models that use disparate property ports. For example, both enthalpy and temperature are needed if one model has a dry bulb temperature port while another has enthalpy. This is exactly the same situation as we encountered when linking models with different units at the ports. The only difference is the constraint equations for fluid links are usually complex thermodynamic relationships rather than simple linear conversion equations.

Figure 13 shows an autolink for moist air using the same graphic representations as in Figure 9. Many commonly needed psychrometric properties are included, as well as flow rate expressed as both volumetric and mass flow rate, making eight sublinks in all. Additionally, there are four psychrometric constraint equations (See Psychrometrics chapter (ASHRAE 1993)):

SpecVol:	Relationship between dry bulb temperature, humidity ratio, specific volume, and pressure.
RelHum:	Relationship between dry bulb temperature, humidity ratio, relative humidity, and pressure.
Enthalpy:	Relationship between dry bulb temperature, humidity ratio, and enthalpy.
Product:	Relationship between mass flow rate, volumetric flow rate, and specific volume.

The detailed representation in the figure, part (a), shows links among the objects representing these relationships. Part (b) shows how this can be implemented with a macro object containing the four psychrometric objects and having a Moist Air Flow (MAF) macro port. Any number of other objects with MAF macro ports could be linked with a moist air link. Moreover, each linked object could use whatever subports of the link were needed in that particular object. We demonstrate this in a SPARK context below.

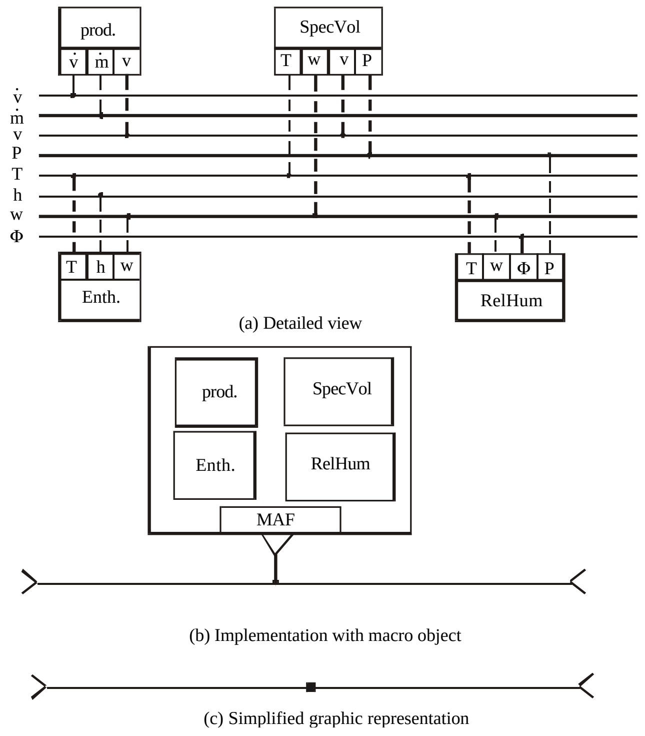


Figure 12: Moist air autolink.

SPARK IMPLEMENTATIONS

Currently, SPARK does not have special syntax for autolinks. Instead, the concept is implemented using the *macro port* feature together with conversion object classes. To demonstrate, consider the system with a diverter, fan and heater shown diagrammatically in Figure 14.

The leftmost object, *div*, is a diverter that splits the inlet moist air stream 1 so that the fraction *f1* leaves through the first branch *af2*, with the remainder going through the second, *af3*. All three of these stream ports are moist air macro ports having all eight properties shown in Figure 13. The fan and heater objects also have inlet and outlet the macro ports of the same structure. Additionally, these objects have various parameters at normal ports.

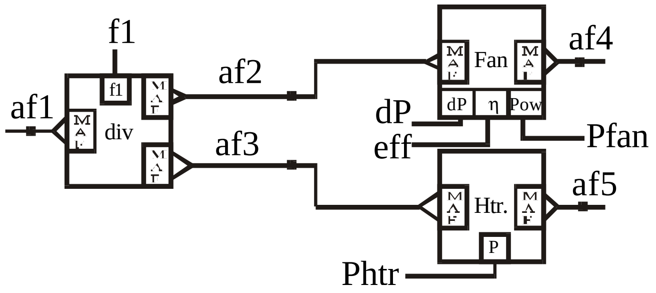


Figure 13: System with moist air links

A SPARK problem file based on Figure 14 is shown below:

```
// Example: moist air autolinks
// Conversion macros
declare airlinkMP aL1, aL2 aL3, aL4, aL5;
declare fanMP f;// Simple fan
declare divMP d; // Diverter
declare heaterMP ht; // Heater
// Inputs
link  effFan "Fan eff." f.eff  input [fraction] ;
link  deltaPFan  "Fan pressure rise"  f.deltaP input [Pa];
link  powHtr  "Heater power" ht.power  input [W];
link  f1 "Fraction of input stream diverted to fan"  d.f1;
link  af1 "Air flow 1" d.AirEnt, aL1.Air input(.TDb),input(.m),input(.w);
// Diverter to fan and heater links
link af2 "Div. to fan Air flow"  div.AirLvg1, f.AirEnt  [MoistAirFlow];
link af3 "Div. to heater" div.AirLvg2,  ht.AirEnt [MoistAirFlow];
// Some reoprts
link powerTot  "Fan Power consumption"  f.powerTot [W]  report;
link af4 "Fan lvg " f.AirLvg, aL4.Air  report(.mAir),report(.TDb),  report(.rh);
link af5 "Heater lvg" ht.AirLvg,aL5.Air  report(.TDb) report(.QAir)  report(.w);
```

The *declare* statements instantiate an *airLinkMP* object for each of the five air streams, *aL1* through *aL5*, as well as each component object. The first four link statements, with the *input* keyword, provide parameters to the component models. The notation *f.eff* means the *eff* port of the *f* object. The next link statement is slightly different since it provides inputs that apply to subports of macro links. This statement means that *AirEnt* macro port of *d* and the *Air* macro port of *aL1* are linked, and that the subports *.TDb*, *.mAir*, and *.w* are all obtained from input.

The link statements for the streams between the diverter and the fan and heater come next. Observe here that a single statement connects all subports. This is a convenience afforded by macro link concept.[2] More important to the theme of this paper, however, is to observe that the diverter model is based on mass flow, temperature, and humidity ratio, while the fan is based on volumetric flow rate, temperature, and humidity ratio, and the heater needs mass flow, enthalpy, and humidity ratio. The presence of the *airlinkMP* objects *aL2* and *aL3* in these linkages provides for automatic conversion to the needed interface variables at each object. This demonstrates the convenience of the macro links plus conversion macro objects, which in SPARK provide us with autolinks.

In a more complete system model, the fan and heater would probably be connected to other components. In the interest of brevity, here we truncate the model after these components. The last three link statements are included merely to allow reporting of some of the properties in the two outlet air streams. Note in particular that we choose to report different properties and flow measures at each object. For example, we report mass flow rate and relative humidity at the fan exit, while reporting volumetric flow and humidity ratio at he heater exit.

As noted, this example uses four different macro classes. The macro class definition for *airlinkMP* is as follows:

```
// Moist air flow autolink
//  Using macro ports
// airlinkMP.cm
//
port Air "Moist air flow" [MoistAirFlow];
   .h  "Enthalpy" noerr [J/kg_dryAir]  init = 25194.2  min = -50300.0 max =   398412.5 ;
   .TDb  "Dry bulb temperature" noerr  [deg_C] init = 20.0   min = - 50.0  max =   95.0 ;
```

[2] ***Note that SPARK infers macro links from linkage to macro ports. There is no explicit macro link syntax.***

```
   .w  "Humidity ratio" noerr  [kg_water/kg_dryAir]  init = .002  min = 0  max = 0.1 ;
   .Patm  "Atmospheric pressure"  [Pa]
   default = 101325;
   .v  "Specific volume" noerr  [m^3/kg_dryAir] ;
   .rh  "Relative humudity" noerr  [fraction]
   init = 0.5 min = 0 max = 0.9 ;
   .mAir  "Air mass flowrate" noerr  [kg_dryAir/s];
   .Qair  "Air volumetric flow" noerr  [m^3/s] ;
declare relhum RH;
declare specvol sv;
declare enthalpy e;
declare safprod sp;

link .Air.QAir sp.c;
link .Air.mAir sp.a;
link .Air.v    sp.b, sv.v;
link .Air.h    e.h ;
link .Air.TDb  e.TDb, sv.TDb, RH.TDb;
link .Air.w    e.w, sv.w, RH.w;
link .Air.PAtm sv.PAtm, RH.PAtm ;
link .Air.rh   RH.rh;
```

The four declare objects are from the global and HVAC libraries that come with SPARK (Sowell and Moshier 1995). They are linked together as indicated in Figure 13, with the wanted properties elevated to subports of the port named Air. The other macro object classes are omitted for brevity, but they have macro ports with the same subports as in *airlinkMP*.

Thus we see that autolinks for property conversions can be implemented with current SPARK syntax. However, we recognize a deficiency in modeling clarity, inasmuch as an autolink is not syntactically explicit. We are considering minor alterations in the modeling language to address this issue. For example, an autoconversion link could be declared as:

```
link AirFlow1 A.Air, B.Air  conversion=airlink ;
```

This compact syntax is possible since an instance of an autolink can appear only in a single statement; hence, there is no need to instantiate it at a higher level. With either of these ideas it is should be clear to a human reader of the code that the macro class *airlink* is playing a conversion role in the link, thus improving clarity.

PRUNING THE COMPUTATION GRAPH

In implementing the autolink concept care must be taken to avoid unnecessary conversions at run-time. We have already noted this problem, citing it as a reason for associating the conversion object with the link rather than the ports of connected objects (see Figure 8). However, a potentially greater inefficiency can arise due to needlessly executing all conversion objects associated with the link. This is of special concern if some of the conversions are complex, and there are many subports in the link, with few of them actually needed. This is often the case with moist air links, for example.

In SPARK, all unnecessary conversion calculations are avoided due to the normal graph-theoretic pre-processing. To see how this works, let us return to the example of Figure 12. After the problem description is parsed, a matching is done between problem variables and objects. After the matching, a “computation graph” is constructed. This is a directed graph in which each vertex represents an equation object, inverted to be a formula for the particular variable selected in the matching step, and its incoming edges represent the variables upon which the formula depends.

The computation graph will clearly show which, if any, of the converted quantities are not used in subsequent calculation, because there will be no leaving edges from their nodes. Therefore there is no need to calculate them and they can be eliminated (pruned) from the computation graph. To discover such nodes one simply traverses the graph backward from vertices specified to be reported. Vertices left unvisited in this traversal are elided. This is routinely done in SPARK processing prior to the solution step. In Figure 12 example, two unnecessary conversions are eliminated by this pruning. A more dramatic example is when both layer models are formulated in SI units, but nonetheless connected with autolinks. While one would perhaps not do this intentionally, the use of libraries to solve diverse problems with a consistent modeling discipline often produces such connections. However, the pruning done in SPARK will simply leave all the conversion objects behind, so at solution time it is as if every model were custom built to interface with its neighbors.

Another issue arises when autolinks are used within macro objects in hierarchical modeling. In this situation an internal autolink introduces a conversion object. If the internal autolink is linked to a macro port of the macro class, and another autolink is connected to this macro port externally, a second conversion be object is introduced. The second conversion object is redundant, over-determining the problem. The problem can be detected as two identical objects with identical links. Such connections can serve no legitimate purpose, so the extra conversion object can be omitted at the parsing stage, or deleted during the graph theoretic analysis. The former is currently done in SPARK.

CONCLUSIONS

In this paper we have presented a view of the automatic conversion linkage concept which we call autolinks. We have shown that this concept is a natural adjunct to the macro object and macro link concepts often used in object based, hierarchical modeling. Indeed, with the SPARK implementation of the latter, no additional syntax is required. That is, autolinks can be expressed entirely in terms of macro objects and macro links. Additionally, it has been shown that the autolinks concept serves equally well for unit conversion and property conversions, such as needed for moist air. We have demonstrated that when used with the normal SPARK solution strategy the autolink concept introduces only the minimum number of conversions at run time. This means that the model developer can use autolinks consistently throughout the problem, mixing models expressed in various units and property sets, while avoiding unneeded convesrion computations.

ACKNOWLEDGEMENTS

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

REFERENCES

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

Bring, A., P. Sahlin, et al. (1992). The Neutral Model Format for Building Simulation, Royal Institute of Technology, Dept. of Building Services Engineering, Stockholm.

Konopasek, M. and S. Jayaraman (1985). “Constraint and Declarative Languages for Engineering Applications: The TK! Solver Contribution.” Proc. IEEE **73**(12): 1791-1806.

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

Sahlin, P. (1988). MODSIM: A Program for Dynamical Modeling and Simulation of Continuous Systems, Swedish Institute for Applied Mathematics, Stockholm.

Sahlin, P. and A. Bring (1991). IDA Solver- A Tool for Building and Energy System Simulation. Building Simulation '91, Nice, France, International Building Performance Simulation Association.

Sahlin, P., A. Bring, et al. (1995). Future Trends of the Neutral Model Format (NMF). Building Simulation '95, Madison, WI, IBPSA.

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

Sowell, E. F. (1998). SPARK Users' Manual. Placentia, CA 92871, Ayres Sowell Associates, Inc.

Sowell, E. F., W. F. Buhl, et al. (1986). A Prototype Object-based System for HVAC Simulation. Proceedings of the Second International Conference on System Simulation in Buildings, Liege, Belgium, Univ. of Liege.

Sowell, E. F. and D. W. Low (1982). ENET: A PC-based Building Energy Simulation System. Proceedings of the IBM Energy Programs Conference, Austin, TX, IBM Real Estate and Construction Division.

Sowell, E. F. and M. A. Moshier (1995). HVAC Component Model Libraries for Equation-based Solvers. Building Simulation '95, Madison, WI, International Building Performance Simulation Association.

Sowell, E. F., G. Silverman, et al. (1981). “Modeling and Optimization of HVAC Systems Using Network Concepts.” ASHRAE Trans. **87**.

Sowell, E. F., K. Taghavi, et al. (1984). “Generation of Building Energy System Models.” ASHRAE Trans. **90** (Pt. 1): 573-86.