Lawrence Berkeley National Laboratory

Recent Work

Title

Symbolic Modeling in Building Energy Simulation

Permalink

<https://escholarship.org/uc/item/9rx2p450>

Authors

Nataf, J.-M.

Winkelmann, F.C.

Publication Date

1994-04-01

Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA

ENERGY & ENVIRONMENT DIVISION

Submitted to Energy and Buildings

Symbolic Modeling in Building Energy Simulation

J.-M. Nataf and F. Winkelmann

April 1994

ENERGY & ENVIRONMENT
DIVISION

LOAN COPY
Circulates
for 4 weeks
Bldg. 50 Library.
Copy 2
LBL-35439

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Symbolic Modeling in Building Energy Simulation

Jean-Michel Nataf* and Frederick Winkelmann

Simulation Research Group
Building Technologies Program
Energy and Environment Division
Lawrence Berkeley Laboratory
Berkeley, CA 94720

April 1994

Abstract

We show how symbolic modeling is used in the Simulation Problem Analysis and Research Kernel (SPARK) for solving complex problems in building energy simulation. After a brief overview of SPARK, we describe its symbolic interface, which reads equations that are entered in symbolic form and automatically generates a program that solves the equations. The application of this method to solving the partial differential equations for two-dimensional heat flow is illustrated.

1. Introduction

Buildings are extraordinarily complex physical systems. To model even a simple house with any degree of accuracy requires solving hundreds of equations representing the interacting processes of heat transfer, operation of heating and cooling equipment, and controls. Of the many computer programs available for building energy simulation, virtually all use traditional programming techniques in which subroutines in a computer language such as FORTRAN or C are written by hand to express and to solve the equations. However, as buildings become even more complex with the introduction of advanced technologies, and as the demand grows for more accurate simulation to support design of better buildings, the traditional approach to simulation has become a deterrent. The main reason for this is that computer code is difficult to write and debug. Using traditional coding methods, only about 5% of development time is spent formulating the underlying mathematical equations that represent the problem to be solved; the remaining 95% is spent on program coding and debugging.

In this paper we describe an alternative to traditional methods. We show how symbolic manipulation and computer algebra techniques in the Simulation Problem Analysis and Research Kernel (SPARK) can simplify and expedite the creation of powerful simulation programs, reducing the development time and leading to code that can be reused for other applications. Symbolic modeling allows equations to be entered by typing them in the same form that they would be written on paper. "Symbolic manipulation" software then interprets the equations, converts them automatically into computer code, and packages them as "equation objects."

* Now at Groupe Informatique et Sciences Energetiques, ENPC, La Courtille-Cedex, Noisy le Grand, France.

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies, Building Systems and Materials Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

But symbolic processing goes further than simply generating equation objects. It can also aggregate two or more equations into a "macro" object that can be stored in a library. These macro objects can then be graphically displayed and connected into networks to form entire, customized programs for simulating building systems of *arbitrary complexity*. The end result is that the program developer is freed to concentrate on modeling rather than on writing and debugging thousands of lines of computer code.

Section 2 gives a brief overview of the SPARK environment. Section 3 describes the use of symbolic manipulation and computer algebra to generate objects, macro objects, networks of macro objects, and entire simulation programs. We conclude in Section 4 with an example application in which symbolic modeling is used to enter and solve the partial differential equations for two-dimensional heat flow.

2. The SPARK Environment

SPARK is an object-oriented environment for developing customized computer models for simulating building energy systems. Given the differential-algebraic equations that represent a physical system, SPARK generates an efficient solution procedure, then implements that procedure in a program that it automatically generates in the C language. The overall organization of SPARK is shown in Figure 1 [Anderson 1986, Buhl 1990].

Figure 1: Configuration of the Simulation Problem Analysis and Research Kernel (SPARK). Shaded boxes are programs; unshaded boxes are files. Ovals show user actions.

The user interacts with SPARK in four basic ways: (1) defining objects (which represent the equations of a physical system), (2) linking objects together to define the simulation problem to be solved, (3) specifying run-time data (parameters and time-varying input data), and (4) specifying desired output. The objects are defined in text files, either as mathematical equations or as component models in Neutral Model Format [Sowell 1989]. These files are processed symbolically producing C-language functions and objects that are stored in libraries.

Problems are defined by interconnecting objects using the graphical user interface, producing a problem specification file in the Network Specification Language (NSL)[Anderson 1986]. From the NSL description, SPARK generates internal data structures based on graphs. Matching and reduction algorithms are used with these graphs to automatically devise an efficient solution algorithm, producing an executable program for each particular problem. This program reads constant and time-varying input data and solves the problem for each time step using Newton-Raphson iteration. The output processor reads the results file and generates graphical displays according to interactive user requests.

SPARK has been successfully used for solving problems encountered in building energy simulation, including air conditioning systems [Buhl 1990], desiccant dehumidification [Nataf 1991], HVAC/lighting interactions [Sowell 1990], coupled natural convection and conduction [Buhl 1990], and in-room convection [Nataf 1993].

3. The Symbolic Interface

An elementary object in SPARK is an algebraic or differential equation (Figure 2). The equation's variables can be thought of as links that can be given values or can be associated with variables in other equations. Linking two objects means that one or more variables are shared by the equations, as illustrated in Figure 3. Thus, SPARK needs to be told what the variables of each equation are and how they are linked with the variables of the other equations. This information is supplied in "object files" that encapsulate all information about each equation.

The SPARK solution method requires functions, in the C language, that solve each equation in terms of each of its variables. For an equation of the form

$$f(x,y,z,\dots)=0,$$

this means that SPARK requires the inverse functions g , h , etc., such that

$$x=g(y,z,\dots), y=h(x,z,\dots), \text{ etc.}$$

Although SPARK users can write the C code for the object and function files by hand, the process is tedious, error prone and time consuming. For example, for an equation with n explicit variables, there are, in general, n C-functions to supply, plus a SPARK object file that tells which functions are associated with which variables. An object corresponding to a physical process or component is usually described by several equations, each of them having an associated object file and routine of function files.

Figure 2: An elementary SPARK object, which represents a single equation. The links of the object are the equation variables.

Figure 3: Linking elementary objects to represent a system of equations. In this example, elementary objects *E1* and *E2* are linked to form a macro object *M1*, which is then linked to *E3*, another elementary object. The links are the variables that are shared among the equations.

A system of equations is a set of elementary objects linked together, as shown in Figure 3. Describing such an object requires creating all of the elementary objects and their associated C-functions and linking the elementary objects into a "macro" object.

To automate the process of defining and linking objects we have developed a symbolic interface to SPARK that is based on MACSYMA, a symbolic manipulation program [MIT 1983]. With this interface, you need only type in the equations for the system that you want to model. The interface consists of a set of commands, whose arguments are the equations. The commands invoke MACSYMA, which in turn creates the appropriate SPARK files.

In the following, we describe how the interface is used to create an *elementary object* (which corresponds to a single equation), a *macro object* (which corresponds to a system of equations), a *dynamic object* (which corresponds to a differential equation), a *dynamic macro object* (which represents a system of differential equations), and a *complete simulation*.

3.1 Generating elementary objects

The simplest object, called an elementary object, is a single algebraic or transcendental equation with no time derivative. It is created with the following command:

makespark (eq, name, badlist)

where *eq* is the equation in symbolic form and *name* is the name of the object. *Badlist* is a list of bad inverses, i.e., a list of variables that you do not want the equation to be solved for because the variables will be input parameters or will exhibit bad numerical properties as iteration variables.

As an example of *makespark*, consider the equation for infrared radiation exchange between two surfaces of temperature T and T_0 :

$$q_{12} = \varepsilon(\theta, \phi)(T^4 - T_0^4)$$

where $\varepsilon(\theta, \phi)$ is the emissivity of the surface as a function of the direction that the radiation leaves the surface. The following command makes this into an object called *my_rad.obj* (Figure 4) that requires the variables θ and ϕ to be input parameters, but retains the ability to calculate the temperatures T and T_0 , or the flux, q_{12} :

makespark (q12=eps(th,phi)*(T^4-T0^4), "my_rad", [th,phi])

Here, *eps* is an external function invoked by the generated C code. After the object *my_rad.obj* is created it is automatically stored in a library for later use as part of larger simulation problems.

As an another example of *makespark* — for the case of a piecewise-defined equation — consider the above equation for radiation exchange, but with a linear approximation for small temperature differences:

$$q_{12} = \varepsilon(\theta, \phi)(T^4 - T_0^4) \text{ if } |T - T_0| > \Delta T$$

$$q_{12} = h_{eq}(\theta, \phi)(T - T_0) \text{ if } |T - T_0| \leq \Delta T$$

where $h_{eq}(\theta, \phi)$ is the equivalent heat transfer coefficient in case of radiative linearization. The command for the creating the new object is:

**makespark ([q12=eps(th,phi)*(T^4-T0^4), (T-T0)^2>delta^2],
[q12=heq(th,phi)*(T-T0), (T-T0)^2<=delta^2], "my_rad2", [th,phi,delta])**

Here, *heq* and *eps* are external C functions present in the function library. Note that *delta* is in the "badlist" since it would not be solved for but would be an input parameter to any problem using this equation.

Figure 4: SPARK macro object representing coupled radiative and convective heat transfer. *Conv* is a dynamic object, which corresponds to a single ordinary differential equation.

3.2 Generating macro objects

A *macro object* is a system of equations. Macro objects are useful for representing complex physical components. For example, a typical HVAC component, such as a heat exchanger, will have several conservation laws to satisfy plus some constitutive behavioral equations. The number of equations depends on how detailed the model is. Describing an entity as a macro object allows the user to treat the entity as a whole — to instantiate it and link it to other objects — without having to worry about its internal details.

The following command creates a macro object along with its subobjects (the elementary objects corresponding to the individual equations) and associated C-language functions:

writemacro (sys, name, listofbadlist)

Here, *sys* is the equation system in symbolic form, *name* is the name of the macro object, and *listofbadlist* is a list of bad inverses for each equation in the system.

The effect of *writemacro* is to put an equation system in network form. This command scans the equations for common variables, states the links between equations with common variables, and generates all the elementary objects associated with the individual equations, along with the C functions that solve the equations for particular variables.

For example, consider the equation system

$$q_{12} = \varepsilon(\theta, \phi)(T^4 - T_0^4)$$

$$q_{12} + q = 0$$

The following command will generate a macro object named *big_rad.obj* that corresponds to this system:

```
writemacro ([ [[q12=eps(th,phi)*(T^4-T0^4)]], [[q12+q=0]] ],
            ["big_rad", "my_rad", ["minus", [q12=in, q=out]]], [[th, phi], []])
```

Here we have specified that θ and ϕ are bad inverses for the first equation; we have specified no bad inverses for the second equation.

3.3 Generating elementary dynamic objects

An elementary dynamic object corresponds to a single ordinary differential equation (ODE). The corresponding SPARK representation actually consists of two equations: the ODE itself (with the derivative given a variable name, for example *xdot*) and the integrator equations, which state that *xdot* is the derivative of *x*, *ydot* is the derivative of *y*, and so on. Thus the SPARK object will actually be a macro object with two subobjects and associated C functions. The command for creating an elementary dynamic object is

```
makedynspark (eq, name, badlist, dynlist)
```

where the first three arguments are the same as those in *makespark* and the last argument, *dynlist*, is a list of pairs *[[x,xdot],[y,ydot],...]* indicating that *xdot* is the derivative of *x*, etc.

Typically, a dynamic object would be a first-order nonlinear differential equation. For example, the following command creates a dynamic object, *conv.obj* (see Figure 4), for the heat transfer equation

$$C \frac{dT_1}{dt} = h_{12}(T_2 - T_1)$$

for the case that the heat capacity, *C*, is always an input parameter:

```
makedynspark (C*T1dot=h12*(T2-T1), "conv", [C], [T1,T1dot])
```

Here *T1dot* is the time derivative of *T1*.

3.4 Generating dynamic macro objects

A dynamic macro object represents a system of differential-algebraic equations. The command for creating this kind of object is

```
writedynmacro (sys, name, listofbadlist, dynlist)
```

where the first three arguments are the same as for *writemacro*, and the fourth argument is a list of lists of the same type as *dynlist* in *makedynspark*.

Most components encountered in building thermal modeling are of this kind; an example is transient heat conduction through a wall discretized into several nodes.

3.5 Generating macro-object networks

Some equation systems have a particularly simple and repetitive form. In heat transfer, for example, the electrical analogy for conductive, convective and radiative transfer leads to equation systems of a simple form:

$$c_k = \sum_{i=1}^N a_{ik} b_i$$

where b is any expression with index i (a vector) and a is any expression with two indices (a matrix). This approach can be used, for example, for conveniently expressing the equations for the radiative interaction between plane surfaces for which the shape factors have two indices and the flux has one index.

The command for generating macro-object networks is

writenericnetmacro (*n*, *name*, *objname*, *expr1*, *expr2*, *badinvlist*)

where n is the number of equations, *name* is the macro name, and *objname* is the name of the elementary object describing the equations. These equations have the form

$$expr1 = \sum_{j=1}^n expr2$$

where *expr1* depends on index k and *expr2* depends on indices k and j .

3.6 Generating a simulation

If the user does not intend to link any objects together by hand in the overall simulation file, and wants everything to be created automatically, then the following syntax can be used:

writesimul (*eq*, *name*)

where *eq* is the overall equation system, including the equations that give the values of the inputs, and *name* is the name assigned to the overall simulation file. The *writesimul* command creates everything that is needed for a SPARK simulation to be ready to run, including the simulation and input files.

3.7 Generating a simulation containing two-dimensional PDE's

SPARK handles systems of algebraic and ordinary differential equations, but has no built-in way to treat partial differential equations (PDE's). Two approaches to handling PDE's in SPARK are illustrated in Figure 5. One approach is to resort to approximate closed-form solutions of the PDE, as determined, for example, by a variational method. The resulting equation is then used to create a SPARK object using the symbolic interface, as described above.

A second approach is to observe that a finite-difference representation of a problem yields a system of differential-algebraic equations that is well suited to treatment with the SPARK object-oriented methods. In the 2-D finite-difference discretization, each elementary bulk domain rectangle can be described by the same object (Figure 6). Furthermore, there are only a few possible configurations for the boundary elements (primarily corners and flat boundaries), which means that only a few types of objects are needed to represent all possible boundary conditions (Figure 6).

The second approach has been implemented in SPARK. It handles second-order PDE's with first-order boundary conditions on 2-D domains of any shape that is regular enough.

For steady state, the command is

writendiff2Dsimul (name, objname, bcname, diffeq, domain, constraint, dx,dy, badlist)

where *name* is the name of the overall simulation file, *objname* is the name of the bulk cell object, *bcname* is the suffix for the name of the boundary condition object (prefixed with *x_*, *y_* or *xy_* depending on whether it is a left/right, top/down or corner boundary), *diffeq* is the PDE, *constraint* specifies the boundary conditions, and *domain* is a 2-D function that is negative inside of the domain and zero at the boundary. The quantities *dx* and *dy* are the spatial discretization steps in the *x* and *y* dimensions, respectively, and *badlist* is the list of variables that we do not want to solve for.

SPARK can therefore handle problems as complex as natural convection in two dimensional enclosures, for example. You need only enter the PDE in symbolic form, together with the boundary conditions and domain geometry. The C code needed to simulate the problem is then automatically generated. An example of using *writendiff2Dsimul* for generating a simulation of 2-D heat conduction in a disk is given in the next section.

Figure 5: Two approaches for handling partial differential equations in SPARK.

Figure 6: Finite difference objects in SPARK.

4. Example of Symbolic Modeling: 2-D Conduction

We consider the simulation of heat conduction in a disk. We take the disk to be a section through an infinitely long rod, so that the heat conduction is in the two dimensions perpendicular to the axis of the rod. The disk has a heat production term in the bulk domain and a Newtonian convection boundary condition. The conduction equation to be solved is

$$k\Delta T + q = \rho c_p \frac{\partial T}{\partial t}$$

where T is the temperature, q is the bulk heat generation rate, k is the thermal conductivity, ρ is the density, and c_p is the specific heat capacity.

The boundary condition on the perimeter of the disk is

$$-k \frac{dT}{dn} = h(T - T_0)$$

where h is the heat transfer coefficient, T_0 is the (constant) ambient temperature, and $\frac{dT}{dn}$ is the normal derivative of the temperature at the boundary.

The commands for generating the simulation for this problem are as follows for a disk with a radius of 1.0m and a discretization of 0.1m in x and y :

```
/*Disk with uniform heat generation, Newtonian convection loss*/
r0: 1.0;
circle1(x,y) := x^2+y^2-r0^2;
ctt: [[-k/r0*(diff(temp,x)*x(x,y)+diff(temp,y)*y(x,y)) = h*(temp(x,y)-temp0)]];
eqdif: [[k*(diff(temp,x,2)+diff(temp,y,2))+q = rho*cp*diff(temp,t)]];
writefindiff2Dsimul ("p2dyn", "p2dynelt", "bcp2dynelt", eqdif, circle1,
    ctt, 0.1, 0.1, [temp0,rho,cp,k,q]);
closefile();
```

The SPARK solution for the disk temperature distribution at 0, 1, 2, and 10 sec is shown in Figure 7 for the following parameters: $k=0.032$ W/(mK), $q=10000$ W/m³, $h=400$ W/(m²K), $\rho=1020$ kg/m³, $c_p=0.24$ J/(kg-K), initial disk temperature = 24C, and ambient temperature = 20C. The simulation that is automatically generated has 1741 objects or links, and 285 iteration variables.

Figure 7: SPARK solution for the x-y temperature distribution of a disk with $10,000 \text{ W/m}^3$ of internal heat generation. The ambient temperature is 20C. Initially the disk temperature is a uniform 24C.

Conclusion

Symbolic modeling in SPARK allows the equations describing physical systems to be entered in a natural form, relieving the model developer from having to write and debug computer code. Symbolic processing reduces model building time, generates component libraries for later reuse, and permits automatic generation of solutions to complex problems, even those involving partial differential equations.

References

- Anderson, J. L. 1986. *A Network Definition and Solution of Simulation Problems*, Lawrence Berkeley Laboratory report no. LBL-21522.
- Buhl, W.F. et al. 1990. *Advances in SPARK*, Proc. Third International Conference on System Simulation in Buildings, Liege, Belgium.
- MIT 1983. *MACSYMA Reference Manual, version 10*, Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.
- Nataf, J.-M. 1987. *Automatic Modeling of Thermal Systems*, Dissertation, Mechanical Engineering Department, Univ. of California, Berkeley.
- Nataf, J.-M. and F. Winkelmann 1991. *Dynamic Simulation of a Liquid Desiccant Cooling System using the Energy Kernel System*, Lawrence Berkeley Laboratory report no. LBL-29610.
- Nataf, J.-M. and E. Wurtz 1993. *Application of the SPARK Environment to 3D Air Flow Problems*, Proc. Building Simulation '93, Adelaide.
- Sowell, E.F., and P. Sahlin 1989. *Neutral Format and Automatic Translation for Building Simulation Submodels*, Proc. Building Simulation '89, Vancouver.
- Sowell, E.F. and J.-M. Nataf 1990. *Radiant Transfer due to Lighting: an Example of Symbolic Model Generation for SPARK*, Proc. Society for Computer Simulation Western Multiconference, San Diego, and Lawrence Berkeley Laboratory report no. LBL-28273.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
TECHNICAL INFORMATION DEPARTMENT
BERKELEY, CALIFORNIA 94720