LBL-29610
UC-350

Lawrence Berkeley Laboratory

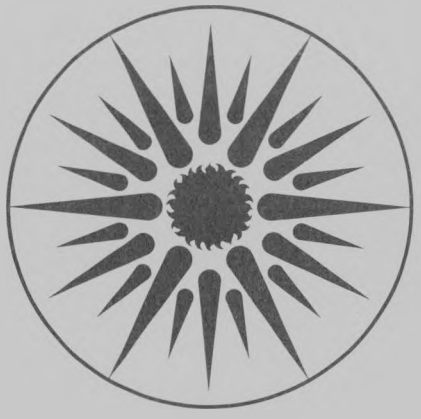
UNIVERSITY OF CALIFORNIA

APPLIED SCIENCE DIVISION

Dynamic Simulation of a Liquid Dessicant Cooling System Using the Energy Kernel System

J.-M. Nataf and F. Winkelmann

February 1991



APPLIED SCIENCE
DIVISION

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

LBL--29610

DE91 010273

DYNAMIC SIMULATION OF A LIQUID DESICCANT COOLING SYSTEM USING THE ENERGY KERNEL SYSTEM

Jean-Michel Nataf and Frederick Winkelmann
Simulation Research Group
Applied Science Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

February 1991

INTRODUCTION

The Energy Kernel System (EKS) is a simulation environment for building energy analysis under development at Lawrence Berkeley Laboratory. EKS is a very flexible, highly modular environment that allows users to create customized models of thermal systems by linking together calculation objects — either defined by the user or obtained from a library — that describe the individual components of the system. A principal departure from other simulation environments is that system models are constructed from submodel objects that are defined without prescribed input or output interfaces, yielding greater modeling flexibility. Also, graph theoretic techniques are employed to determine the solution sequence, including reduction of the iterative problem size at each time step.

To demonstrate the use of EKS for modeling complex physical systems, we present in this paper a dynamic EKS simulation of a hybrid liquid desiccant cooling system. We show (1) how EKS calculation objects are generated automatically using MACSYMA (MIT, 1983), given the basic algebraic and differential equations for the system; (2) how EKS objects are linked into macro objects that describe system components; and (3) how macro objects are linked together to form a mathematical network representing the entire system. Finally, we show graphically the numerical results of running a time-dependent simulation of the system.

This work was supported 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 under Contract No. DE-AC03-76SF00098.

THE EKS ENVIRONMENT

The EKS simulation environment is based on the intuition that:

(a) there should be a single "sub-model" for each particular component,

(b) the overall system model should be defined once, yet be capable of solving any well-posed problem involving the system variables, and

(c) the environment software should select the appropriate solution sequence, including necessary iteration, in a manner transparent to the modeler.

The underlying principles making this realizable for static (i.e., algebraic) systems have been described by Sowell, Buhl, Erdem & Winkelmann (1986), implemented by Anderson (1986), and extended to handle dynamic (i.e., differential-algebraic) systems (Sowell and Buhl 1988). These ideas are described in terms of the object-oriented paradigm in Sowell, Buhl and Nataf (1989) and Buhl et al (1990).

Briefly summarized, EKS is a nonlinear equation solver with automatic equation system reduction and with an object-oriented interface. EKS manipulates objects that are equations and macro objects, which are collections of equations. The task of the user is to generate the objects and their related software equivalent, and to link them together in the appropriate way to take into account the variables common to different objects.

In this way, large systems are modeled by a connection of simple component models, together with descriptions of how the individual components are related. The promise of this approach is that the resulting description of a large system is essentially a schematic of the corresponding physical system, and is constructed in a similar manner to the physical system.

Step by step, then, the procedure that EKS users follow in setting up and running simulation is as follows:

(1) Draw the system schematic showing the physical components of the system and how they are connected.

(2) Write the mathematical equations, such as energy balances, mass balances, etc., that describe the system. These equations are the basic objects that EKS manipulates.

(3) Run the EKS MACSYMA preprocessor to generate the **C** code and associated functions for the equations (objects) in (2).

(4) Using the EKS Network Specification Language (NSL), link the objects in (3) into macro objects (sets of equations) that describe system components.

(5) Using NSL, link the macro objects together into a network describing the entire system.

(6) Specify input variables, starting values, start time, stop time, time step, etc.

(7) Run the simulation.

(8) Plot results.

If macro objects for the system components already exist in the EKS library, then only steps (5) through (8) are necessary. In the following sections we illustrate the above steps in an EKS solution of a hybrid liquid desiccant cooling system.

A diagram of the current EKS environment is shown in Fig. 1. The user interacts with the system in four basic ways: defining objects (e.g., component models); defining

problems by linking objects together; specifying run-time data; and specifying desired output. The objects are defined in text files, either as mathematical equations or as component models in Neutral Model Format (Sahlin and Sowell, 1989). These files are processed symbolically with programs written in MACSYMA, producing C language functions and objects that are stored in libraries. Problems are defined by interconnecting objects using the interactive graphical user interface, producing a problem specification file in NSL. The nucleus or kernel is the dynamic Simulation Problem Analysis Kernel (SPANK) program system (Sowell and Buhl 1988). It works from the NSL description, generating internal data structures based on graphs. Matching and reduction algorithms are employed with these graphs to automatically devise an efficient solution algorithm, producing an executable program for a particular problem. This program reads constant and time-varying data from files, producing the problem solution. The output processor reads the result file and generates graphical displays according to interactive user requests.

PROBLEM DESCRIPTION

We consider the hybrid liquid desiccant system shown schematically in Fig. 2. The fluid used is a solution of lithium chloride in water. A complete description of the system is given in Sick (1986) and Buschulte (1984). The system provides cool, dry air to a space. It contains an interchanger, a heater and a cooler (all modeled with the LMTD method), a regenerator and a conditioner (both modeled with a Kathabar equation), and two sumps, one of which is massive and, therefore, dynamic.

Quoting Sick (1986), "Precooled desiccant solution flows counter-currently to the air stream through the conditioner where it absorbs water vapor and cools down the air only to the desired set temperature. The water taken from the air goes into the liquid desiccant solution. In order to maintain its concentration, the salt solution is pumped to a regenerator. The process in the regenerator is reverse to that in the conditioner. Return air from the building absorbs water from the preheated solution which becomes more concentrated and is pumped back to the conditioner. The conditioner and the regenerator are connected by a heat exchanger (interchanger). The hot solution leaving the regenerator heats up the cooler desiccant coming from the conditioner. Thus, the solution entering the conditioner cycle is precooled, while the solution flow to the regenerator is preheated."

In this exercise we have not modeled controls (such as turning off the conditioner when the regenerator cannot keep up with it).

SYSTEM EQUATIONS

The system schematic for this problem is shown in Fig. 2, which also indicates the system variables. The equations of the system are given below. Here, as in Fig. 2, W stands for mass flow, x for solution salt concentration, H for humidity ratio, i for specific enthalpy, T for temperature, and m for the mass of the solution in the regenerator sump. We have only listed the primary equations; the thermodynamic state equations are not shown.

Conditioner water mass balance:

$$W_4H_4+W_6(1-x_6)=W_4H_5+W_{40}(1-x_{40})$$

Conditioner salt mass balance:

$$W_6x_6=W_{40}x_{40}$$

Conditioner energy balance:

$$W_6i_6(T_6,x_6)+W_4i_4(T_4,H_4)=W_{40}i_{40}(T_{40},x_{40})+W_4i_5(T_5,H_5)$$

Conditioner Kathabar relations:

$$H_5=H(T_5,x_6)$$

$$T_5-T_6=K_C[i_4(T_4,H_4)-i_5(T_5,H_5)]$$

Conditioner sump mass and energy balance:

$$W_{15}+W_{40}=W_6+W_9$$

$$W_{15}x_3+W_{40}x_{40}=(W_6+W_9)x_6$$

$$W_{15}i_{15}(T_{15},x_3)+W_{40}i_{40}(T_{40},x_{40})=(W_6+W_9)i_8(T_8,x_6)$$

Solution cooler heat transfer:

$$UA_c\Delta T_{lm,c}=W_{16}c_{p,w}(T_{14}-T_{13})$$

where $$\Delta T_{lm,c}=\frac{(T_8-T_{14})-(T_6-T_{13})}{\ln\frac{T_8-T_{14}}{T_6-T_{13}}}$$

Solution cooler energy balance:

$$W_{16}[i_{13}(T_{13})-i_{14}(T_{14})]=W_6[i_6(T_6,x_6)-i_8(T_8,x_6)]$$

Interchanger relations:

$$UA_i\Delta T_{lm,i}=W_{15}c_{p,s}(T_7-T_{15})$$

where $$\Delta T_{lm,i}=\frac{(T_7-T_9)-(T_{15}-T_8)}{\ln\frac{T_7-T_9}{T_{15}-T_8}}$$

$$W_9[i_9(T_9,x_6)-i_8(T_8,x_6)]=W_{15}[i_7(T_7,x_3)-i_{15}(T_{15},x_3)]$$

Desiccant heater relations:

$$UA_h\Delta T_{lm,h}=W_{12}c_{p,w}(T_{11}-T_{12})$$

where $$\Delta T_{lm,h}=\frac{(T_{11}-T_3)-(T_{12}-T_7)}{\ln\frac{T_{11}-T_3}{T_{12}-T_7}}$$

$$W_{12}[i_{11}(T_{11})-i_{12}(T_{12})]=W_3[i_3(T_3,x_3)-i_7(T_7,x_3)]$$

Regenerator equations:

$$W_1H_1+W_3(1-x_3)=W_1H_2+W_{41}(1-x_{41})$$

$$W_3x_3=W_{41}x_{41}$$

$$W_3i_3(T_3,x_3)+W_1i_1(T_1,H_1)=W_{41}i_{41}(T_{41},x_{41})+W_1i_2(T_2,H_2)$$

$$H_2=H(T_2,x_3)$$

$$T_2-T_3=K_R[i_1(T_1,H_1)-i_2(T_2,H_2)]$$

Regenerator sump dynamic equations:

$$\frac{dm}{dt}=(W_9+W_{41})-(W_3+W_{15})$$

$$\frac{d(mi_7)}{dt}=(i_9W_9+i_{41}W_{41})-(W_3+W_{15})i_7$$

OBJECT GENERATION

In EKS, the objects corresponding to the above equations are automatically generated using MACSYMA, a symbolic language for manipulating equations. We have written a program in this language that allows the user to enter equations or systems of equations in a natural form. MACSYMA then generates all the object files, macro object files and C function files required by EKS (Sowell and Nataf, 1990).

These files are generated by repeated use of the command **makespank(eq,name)**, which creates the object *name.obj* and associated functions corresponding to equation *eq*. Equation *eq* can be either a single relationship covering the full range of its variables, or a piecewise-defined relationship consisting of different equations for different ranges of its variables.

The set of **makespank** commands for the present problem is given in Fig. 3. We note for example that the object *rc_frac_cons22*, which is created by the second command line

*makespank(mass_in*frac_in=mass_out*frac_out,"rc_frac_cons22",[]),*

can be used for the salt mass balance equation for both the conditioner ($W_6x_6=W_{40}x_{40}$) and the regenerator ($W_3x_3=W_{41}x_{41}$).

MACRO OBJECT GENERATION

We next generate the macro objects, which represent the sets of equations that correspond to individual physical components of the system. This is done by using the EKS Network Specification Language to link objects (each of which represents a single equation) together. An example of this process is given in Fig. 4, which shows how the macro object for a heat exchanger is generated by linking the objects *x_enth_cons22h* (enthalpy conservation) and *newcross_lmtd* (LMTD equation) that were created by MACSYMA in Fig. 3. Because linking of objects by hand is tedious and error prone, we are developing a graphical interface to automate this process. We also note that objects and macro objects, once generated, can be stored in a library and reused later, so that they do not have to be regenerated each time they are needed.

SYSTEM NETWORK GENERATION

The next step is to link the component macro objects together into a network that follows the system schematic. Again, the Network Specification Language is used, as shown in Fig. 5.

In NSL, the command **declare** is used to instantiate (create particular instances of) objects. For example, in Fig. 5, the lines

```
declare    exchanger  cooler;
declare    exchanger  heater;
```

indicate that the macro object "exchanger" is used for both the cooler and heater components in Fig. 2.

The NSL command **input** is used to specify input variables. For example,

```
input   t1(regenerator.temp_in1)[T]
```

specifies that the incoming air temperature for the regenerator will be input by the user.

The **link** command identifies variables (unknowns) that are shared by two or more objects (or macro objects). For example,

```
link   t3(heat.temp_out1,regenerator.temp_in2)[tsump]
```

specifies that the temperature *heat.temp_out1* of the LiCl solution leaving the "heater" component (Fig. 2) is the same as the temperature *regenerator.temp_in2* of the solution entering the "regenerator" component. The user has chosen to label this temperature "t3".

The linking process, which has been done by hand in this example, is automated by the previously-mentioned graphical interface.

THE EKS SIMULATION

The final step is to supply a file with numerical values of input quantities such as weather data, setpoint values, system parameters, starting values, time step, start and stop time, etc. At this point the user can run the actual simulation of the system network.

For the present problem a 9-hour run period was selected, with a time step of 1.5 minutes. All input variables were chosen to be constants (with values as shown in Fig. 2) except for $T4$, the temperature of the air entering the conditioner. This temperature followed a weather profile that increased from 76F to 97F and then fell to 81F.

RESULTS

The simulation results are shown in Fig. 6. Some noteworthy trends are evident:

(1) The humidity, *H5*, of the conditioned air varies from 0.0062 to 0.0052, and is less than the humidity $H4 = 0.0093$ of the incoming air. This shows that the conditioner is drying the incoming air, as is it supposed to.

(2) The humidity, *H2*, of the exhaust air from the regenerator varies from 0.027 to 0.022, and is greater than the return air humidity $H1 = 0.0093$. This indicates that the regenerator is removing water from the salt solution, as is intended.

(3) The rapid variation during the first half hour in variables associated with the regenerator loop, such as $T7$ and $W41$, is a system transient due to the startup dynamics of the regenerator sump.

(4) The cooling power, *del*, of the conditioner follows the $T4$ profile, as expected.

(5) The exhaust air humidity of the regenerator, *H2*, and the conditioner, *H5*, decrease with time.

(6) The mass, *m*, of the solution in the regenerator sump decreases, which means that the amount of water removed from the solution by the regenerator is greater than the amount of water added to the solution by the conditioner.

(7) The leaving cold water temperature, *T14*, from the cooler varies inversely with *T4*, whereas the leaving hot water temperature, *T12*, from the heater is nearly constant.

(8) The regenerator exhaust air temperature, *T2*, and the solution temperatures in the regenerator loop (*T3*, *T7*, and *T41*) increase slowly after an initial sharp drop due to the sump startup transient (see (3), above).

(9) The solution temperatures in the conditioner loop (*T6*, *T8*, and *T15*) track the *T4* variation, whereas the inlet solution temperature is almost constant.

(10) The cold water flow, *W16*, in the cooler tracks *T4*. On the other hand, the conditioner flow, *W40*, and the regenerator flow, *W41*, increase steadily in order to compensate for the loss of solution water by the system (see (6), above).

(11) The net loss of circulating water causes a steady rise in salt concentration (*X3*, *X6*, *X40*, *X41*).

CONCLUSION

We have demonstrated that complex EKS objects and macro objects can be created automatically with available symbolic manipulation tools, and that the EKS can be used to generate a simulation program for modeling a complex real-world problem.

At this point it is important to stress that the input-output-free nature of EKS allows the user to set up and run variants of the problem with little extra effort. For example, it is easy to define new problems that involve changing an input variable into an unknown and *vice-versa*, as long as the problem remains well posed, i.e., as long as the number of unknowns equals the number of equations. This is done by interchanging **input** and **link** commands in Fig. 4. EKS will then automatically generate and solve the new simulation problem. Finally, the inherent modularity of EKS makes it possible to model part or all of the base problem in greater (or lesser) detail by replacing one or more macro objects. For example, in the current problem we used the LMTD method for the heat exchanger macro object. This could be replaced by a more detailed macro object based on a finite difference approach, for instance, with no change to the rest of the problem.

FUTURE WORK

The current EKS program, although already capable of handling a wide range of simulation problems in HVAC analysis, should be considered a prototype. Work is under way or is being planned to significantly expand the usability and robustness of the program. This work includes:

(1) completion of the interactive graphical editor, based on X-Windows, to facilitate retrieval, storage, display, editing, and linking of objects;

(2) expansion of the existing object library to encompass a range of commonly-used HVAC system components;

(3) extension and refinement of the Neutral Model Format, which is particularly important as a way of exchanging component models between EKS and other simulation environments;

(4) replacement of the present Network Specification Language with a new "Component Description Language" that will permit object-oriented specification of integration methods and variable time stepping (Moshier and Sowell, 1990);

(5) incorporation of methods to automatically partition a network into smaller subproblems in order to improve convergence and reduce execution time;

(6) development of an interactive data manager to facilitate data input and results display.

(7) in a parallel effort to EKS development, incorporation of EKS object-oriented techniques into DOE-2 to produce component-based SYSTEMS and PLANT subprograms; the resulting new program, DOE-3, will allow users to easily model new HVAC technologies while retaining the powerful DOE-2 LOADS calculation.

REFERENCES

Anderson, J. L. 1986. "A Network Definition and Solution of Simulation Problems," LBL Report No. LBL-21522.

Buhl, W.F. et al. 1990. "The U.S. EKS: Advances in the SPANK-based Energy Kernel System," *Proceedings of the Third International Conference on System Simulation in Buildings*, Liege, Belgium, and LBL Report No. LBL-29419.

Buschulte, K.T. 1984. "Analysis of Hybrid Liquid Desiccant Systems", M.S Thesis, Department of Chemical Engineering, University of Wisconsin Madison.

MIT 1983. *MACSYMA Reference Manual, version 10,* Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

Sahlin, P. and Sowell, E.F. 1989. "Neutral Format and Automatic Translation for Building Simulation Submodels." *Proceedings of Building Simulation '89,* Vancouver, and LBL Report No. LBL-28274.

Sick, F. 1986. "Analysis of the Seasonal Performance of Hybrid Liquid Desiccant Cooling Systems", M.S. Thesis, Department of Chemical Engineering, University of Wisconsin Madison.

Sowell, E.F., W.F. Buhl, A.E. Erdem, and F.C. Winkelmann 1986. "A Prototype Object-based System for HVAC Simulation." *Proceedings of the Second International Conference on System Simulation in Buildings*, Liege, Belgium, and LBL Report No. LBL-22106.

Sowell, E.F. and W.F. Buhl 1988. "Dynamic Extension of the Simulation Problem

Analysis Kernel (SPANK)." *Proceedings of the USER-1 Conference*, Ostend, Belgium, and LBL Report No. LBL-26262.

Sowell, E.F., W.F. Buhl and J.M. Nataf 1989. "Object-oriented Programming, Equation-based Submodels, and System Reduction in SPANK." *Proceedings of Building Simulation '89,* Vancouver, British Columbia, and LBL Report No. LBL-28272.

Sowell, E.F. and Moshier, M.A. 1990. "Specifying Dynamic Models in the Simulation Problem Analysis Kernel." *Proceedings of the Society for Computer Simulation Western Multiconference*, San Diego, and LBL Report No. LBL-28275.

Sowell, E.F. and Nataf, J.M. 1990. "Radiant Transfer due to Lighting: an Example of Symbolic Model Generation for SPANK." *Proceedings of the Society for Computer Simulation Western Multiconference*, San Diego, and LBL Report No. LBL-28273.

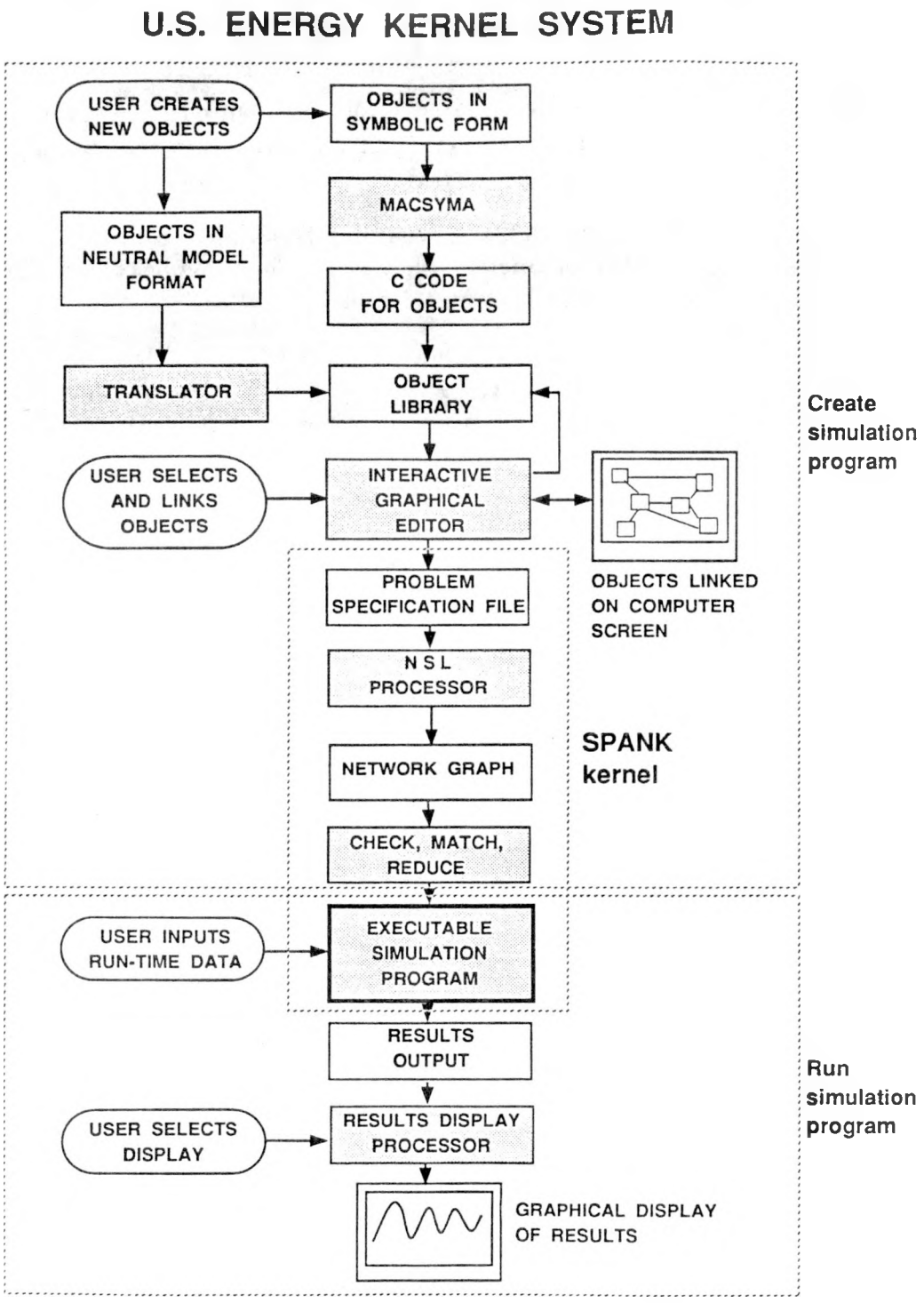


Figure 1. Configuration of the Energy Kernel System (EKS). Shaded boxes are programs; unshaded boxes are files. Ovals show user actions. From objects representing the mathematical equations of a physical system, EKS creates an executable program that can be run to determine the dynamic behavior of the system.

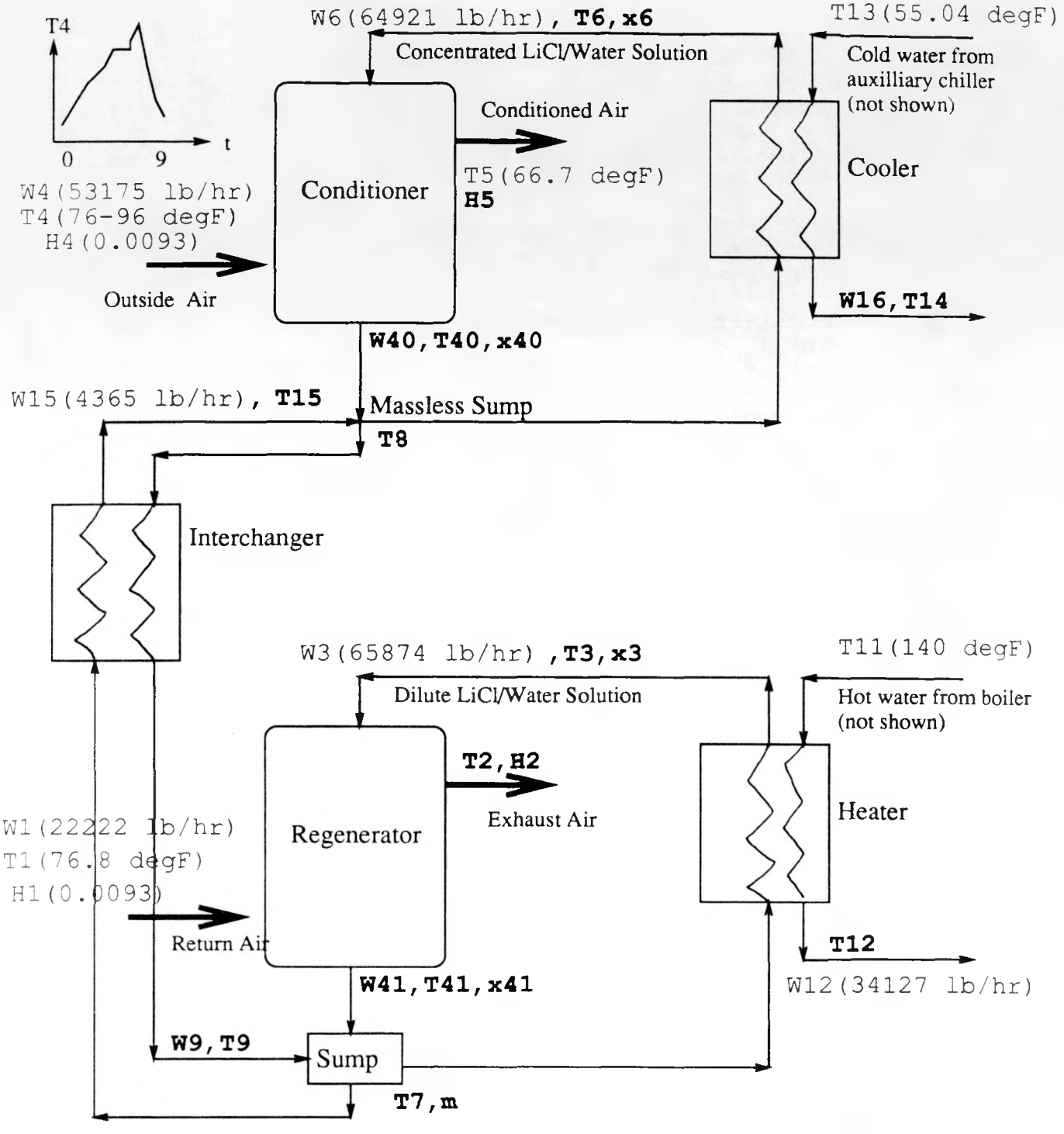


Figure 2. Schematic of a hybrid liquid desiccant cooling system. Unknown variables are shown in boldface. User-specified input variables are shown in lighter type, with values in parentheses.

```
/*************************************************/
/*MACSYMA FUNCTION CALLS TO GENERATE SPANK OBJECTS*/
/*************************************************/

makespank(mass_1*frac_in1+mass_in2*(1.-frac_in2)
        =mass_1*frac_out1+mass_out2*(1.-frac_out2)
        ,"rc_mass_cons22",[])$

makespank(mass_in*frac_in=mass_out*frac_out
        ,"rc_frac_cons22",[])$

makespank(mass_1*enth_in1+mass_in2*enth_in2
        =mass_1*enth_out1+mass_out2*enth_out2,
        "rc_enth_cons22h",[])$

makespank(temp_out1-temp_in2=f*(enth_in1-enth_out1),
        "kath2h",[])$

makespank(mass_in1+mass_in2=mass_out1+mass_out2,
        "s_mass_cons22",[])$

makespank(mass_in1+mass_in2=mass_out1+mass_out2+mdot,
        "ds_mass_cons22",[])$

makespank(mass_in1*frac_in1+mass_in2*frac_in2
        =(mass_out1+mass_out2)*frac_out,
        "s_frac_cons22",[])$

makespank(mass_in1*enth_in1+mass_in2*enth_in2
        =(mass_out1+mass_out2)*enth_out,
        "s_enth_cons22h",[])$

makespank(mass_in1*enth_in1+mass_in2*enth_in2
        =(mass_out1+mass_out2)*enth_out+menthoutdot,
        "ds_enth_cons22h",[])$

makespank(m*enth_outdot
        =mass_in1*enth_in1+mass_in2*enth_in2-(mass_in1+mass_in2)*enth_out,
        "ds2_enth_cons22h",[])$

makespank(mass_2*(enth_in2-enth_out2)
        =mass_1*(enth_out1-enth_in1),
        "x_enth_cons22h",[])$

makespank(h=cp*t,"hcpt",[])$

makespank(ua*((temp_in1-temp_out2)-(temp_out1-temp_in2))/
        log((temp_in1-temp_out2)/(temp_out1-temp_in2))
        =mass_in2*cp2*(temp_out2-temp_in2),
        "newcross_lmtd",[])$

makespank(del=w*(i_in-i_out),"coolpower",[w,i_in,i_out])$
```

Figure 3. MACSYMA function calls that generate the EKS objects for the desiccant cooling system problem.

```
/*****************************************/
/*SPANK FILE FOR THE EXCHANGER MACRO OBJECT*/
/*****************************************/

/* Spank macro object file exchanger.obj */
/* This file contains an .obj file needed to model
 * the hybrid liquid dessicant cooling system described
 * by the M.S. thesis of Friedrich Sick (University of Wisconsin
 * -Madison) */
/* It makes use of the files contained in dessic.c */
/* Can be used for a heater or a cooler*/


/*
 * exchanger object
 */

/* LMTD formula (Logarithmic Mean Temperature Difference)
 * Energy balance
 */

macro
        declare newcross_lmtd   l;
        declare x_enth_cons22   e;

        link    ua(l.ua)
        link    cp(l.cp2,e.cp)

        link    mass_1(e.mass_1)
        link    mass_2(e.mass_2,l.mass_in2)
        link    temp_in1(e.temp_in1,l.temp_in1)
        link    temp_out1(e.temp_out1,l.temp_out1)
        link    temp_in2(e.temp_in2,l.temp_in2)
        link    temp_out2(e.temp_out2,l.temp_out2)
        link    frac_1(e.frac_1)[x5]
        link    enth_in1(e.enth_in1)
        link    enth_in2(e.enth_in2)
        link    enth_out1(e.enth_out1)
        link    enth_out2(e.enth_out2)
```

Figure 4. The EKS file in which objects are linked together to form the macro object representing a heat exchanger.

```
/*******************************************************/
/*SPANK SIMULATION FILE FOR DESICCANT COOLING PROBLEM*/
/*******************************************************/

/* Spank simulation file dyn_desicc1.ps */

/* This file contains the links and inputs necessary
 * for the matching of the various local variables in
 * the hybrid liquid desiccant cooling system problem.
 */
/*Buschulte's Problem+Sick's Thesis, Wisconsin Madison*/

declare regcond          conditioner;
declare regcond          regenerator;
declare iexchanger       interchanger;
declare exchanger        cooler;
declare exchanger        heater;
declare sump22           condsump;
declare part_dyn_sump22  regsump;
declare prod             mx3p;
declare cpbctf           cpbctfi;
declare coolpower        cool;

/*Inputs*/
input kc(conditioner.f)[kathabar]        /*Kathabar constant for conditioner*
input kr(regenerator.f)[kathabar]        /*Kathabar constant for regenerator*
input uai(interchanger.ua)[UA]  /*UA value of interchanger*/
link cpi(interchanger.cp,cpbctfi.cpbc)[hw]      /*Specific heat of
                                        LiCl-water mixture in interchanger*/
input uac(cooler.ua)[UA]         /*UA value of cooler*/
input cpc(cooler.cp)[hw]         /*Specific heat of cold water in cooler*/
input uae(heater.ua)[UA]         /*UA value of heater*/
input cp2(heater.cp)[hw]         /*Specific heat of warm water in heater*/
input t1(regenerator.temp_in1)[T]        /*Temperature of incoming air
                                        in regenerator*/
input t4(conditioner.temp_in1)[T]        /*Temperature of incoming air
                                        in conditioner*/
input t5(conditioner.temp_out1)[T]       /*Temperature of cool dry air
                                        coming out of conditioner into
                                        the conditioned space*/
input t11(heater.temp_in2)[T]    /*Temperature of incoming warm water
                                 in heater*/
input t13(cooler.temp_in2)[T]    /*Temperature of incoming cold water
                                 in cooler*/
input h1(regenerator.frac_in1)[w]        /*Humidity of incoming air in
                                        regenerator*/
input h4(conditioner.frac_in1)[w]        /*Humidity of incoming air in
                                        conditioner*/
input w1(regenerator.mass_1)[bigmw]      /*Mass flow of air
                                        through regenerator*/
input w3(heater.mass_1,regenerator.mass_in2,regsump.mass_out2)[bigmw]
                          /*Mass flow of LiCl-water diluted mixture
                          through heater into regenerator*/
input w4(conditioner.mass_1,cool.w)[bigmw]       /*Mass flow of
                          incoming air through conditioner*/
```

Figure 5. The EKS file that generates the desiccant cooling system network. The network is formed by linking together the macro objects that represent individual system components (conditioner, interchanger, regenerator, etc.) and by assigning input variables.

```
input w6(cooler.mass_1,conditioner.mass_in2,condsump.mass_out2) [bigmw]
                        /*Mass flow of LiCl-water concentrated solution
                        through cooler into conditioner*/
input w12(heater.mass_2) [bigmw] /*Mass flow of warm water through heater*/
input w15(condsump.mass_in2,interchanger.mass_2,regsump.mass_out1) [bigmw]
                        /*Mass flow of LiCl-water mixture out of
                        regenerator sump into conditioner
                        sump through interchanger*/


/*Regenerator Sump exit conditions*/
link t7(heater.temp_in1,interchanger.temp_in2, regsump.temp_out) [tsump]
link x3(heater.frac_1,regenerator.frac_in2,interchanger.frac_2,
        condsump.frac_in2,regsump.frac_out,mx3p.in2) [x5]

/*Unknowns*/
link t3(heater.temp_out1,regenerator.temp_in2) [tsump]
link t12(heater.temp_out2) [t12]
link h2(regenerator.frac_out1) [w]
link t2(regenerator.temp_out1) [tsump]
link w41(regenerator.mass_out2,regsump.mass_in1) [bigmw]
link x41(regenerator.frac_out2,regsump.frac_in1) [x5]
link t41(regenerator.temp_out2,regsump.temp_in1) [tsump]
link w40(conditioner.mass_out2,condsump.mass_in1) [bigmw]
link x6(cooler.frac_1,conditioner.frac_in2,condsump.frac_out,
        interchanger.frac_1, regsump.frac_in2,cpbctfi.f) [x5]
link h5(conditioner.frac_out1) [w]
link x40(condsump.frac_in1,conditioner.frac_out2) [x5]
link w9(interchanger.mass_1,condsump.mass_out1,regsump.mass_in2) [bigmw]
link t6(cooler.temp_out1,conditioner.temp_in2) [tsump]
link t40(conditioner.temp_out2,condsump.temp_in1) [tsump]
link t15(interchanger.temp_out2,condsump.temp_in2) [tsump]
link t9(interchanger.temp_out1,regsump.temp_in2) [tsump]
link t8(condsump.temp_out,cooler.temp_in1,interchanger.temp_in1,cpbctfi.t)
                                                        [tsump]
link w16(cooler.mass_2) [bigmw]
link t14(cooler.temp_out2) [t14]

/*Enthalpies. Careful, since fraction and temperature transmitted
 *---> enthalpy transmitted too. So beware of linking enthalpies together
 *if temps and fracs already linked*/
/*Essentially we here just give names...*/
link i1(regenerator.enth_in1) [h]
link i2(regenerator.enth_out1) [h]
link i3(/*regenerator.enth_in2,*/heater.enth_out1) [hw]
link i4(conditioner.enth_in1,cool.i_in) [h]
link i5(conditioner.enth_out1,cool.i_out) [h]
link i6(cooler.enth_out1/*,conditioner.enth_in2*/) [hw]
link i7(/*interchanger.enth_in2,heater.enth_in1,*/regsump.enth_out) [hw]
link i8(condsump.enth_out/*,cooler.enth_in1,interchanger.enth_in1*/) [hw]
link i9(interchanger.enth_out1/*,regsump.enth_in2*/) [hw]
link i11(heater.enth_in2) [hw]
link i12(heater.enth_out2) [hw]
link i13(cooler.enth_in2) [hw]
link i14(cooler.enth_out2) [hw]
link i15(interchanger.enth_out2/*,condsump.enth_in2*/) [hw]
```

Figure 5 (cont.)

```
link i40(conditioner.enth_out2/*,condsump.enth_in1*/) [hw]
link i41(regenerator.enth_out2/*,regsump.enth_in1*/) [hw]

link del(cool.del) [qq]

link m(regsump.m,mx3p.in1) [desiccm]
link mdot(regsump.mdot)

link mi7(regsump.menthout) [mi7]
link mi7dot(regsump.menthoutdot)

input mx3(mx3p.product) [desiccm]

history m_hist(regsump.m_hist)
history mi7_hist(regsump.menthout_hist)

input dt(regsump.dt) [TIME]
```

Figure 5 (end)

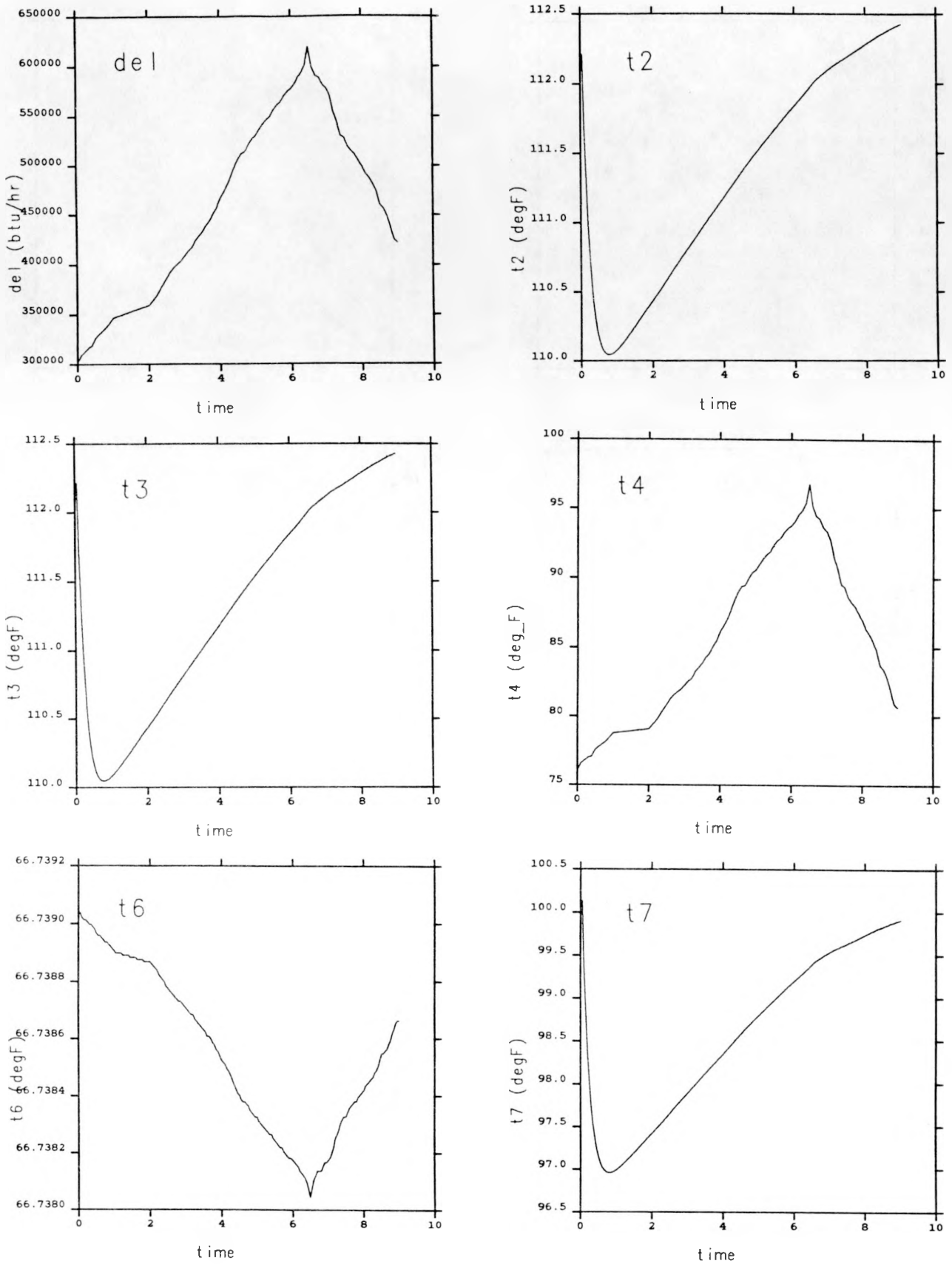


Figure 6. EKS simulation results for the dynamic behavior of the desiccant cooling system shown in Fig. 2. A 9-hour run period was modeled with a time step of 1.5 minutes. Key: w = mass flow, x = salt concentration, h = humidity ratio, i = specific enthalpy, t = temperature, m = mass of sump, del = cooling power of conditioner.

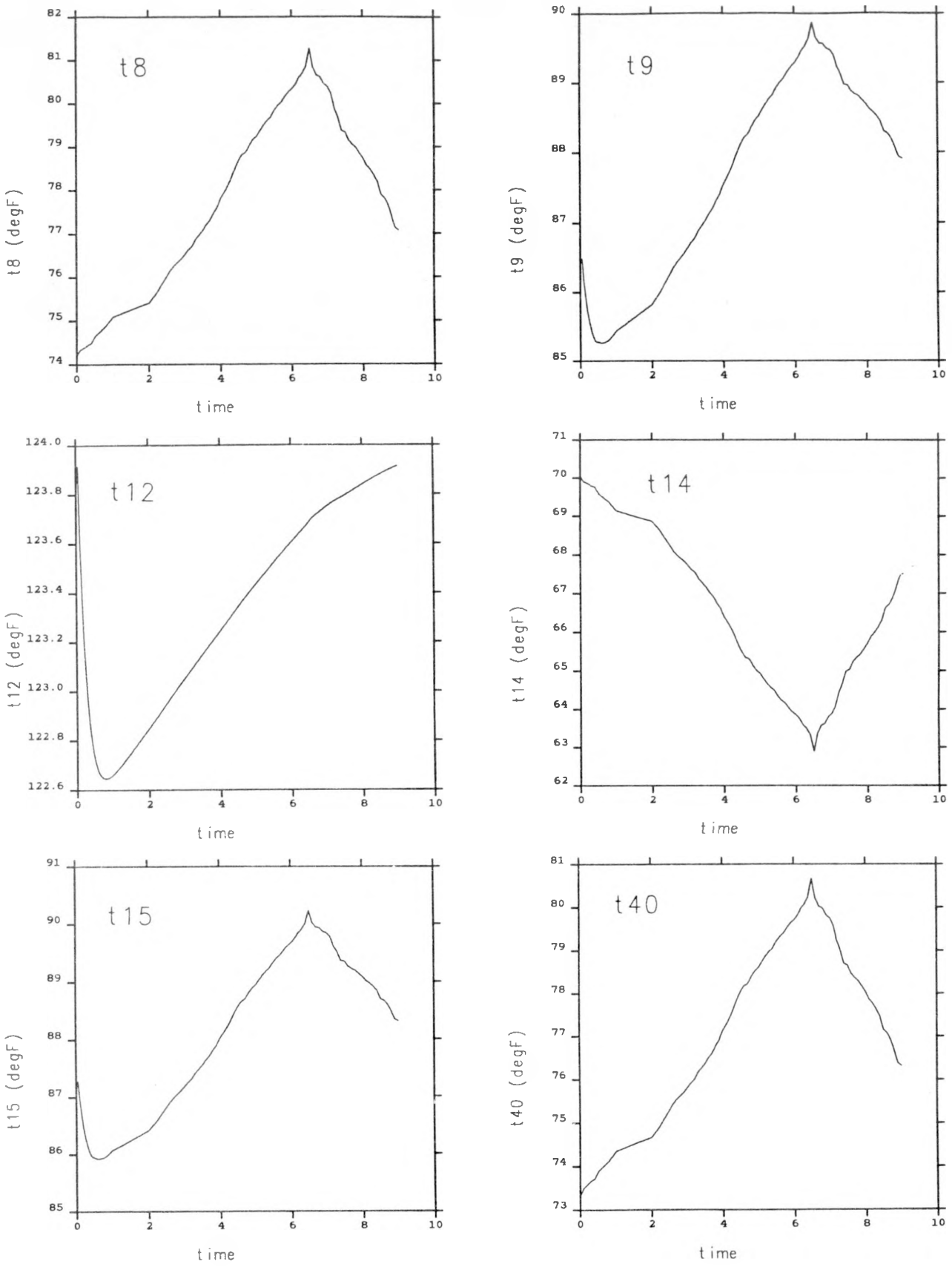


Figure 6 (cont.)

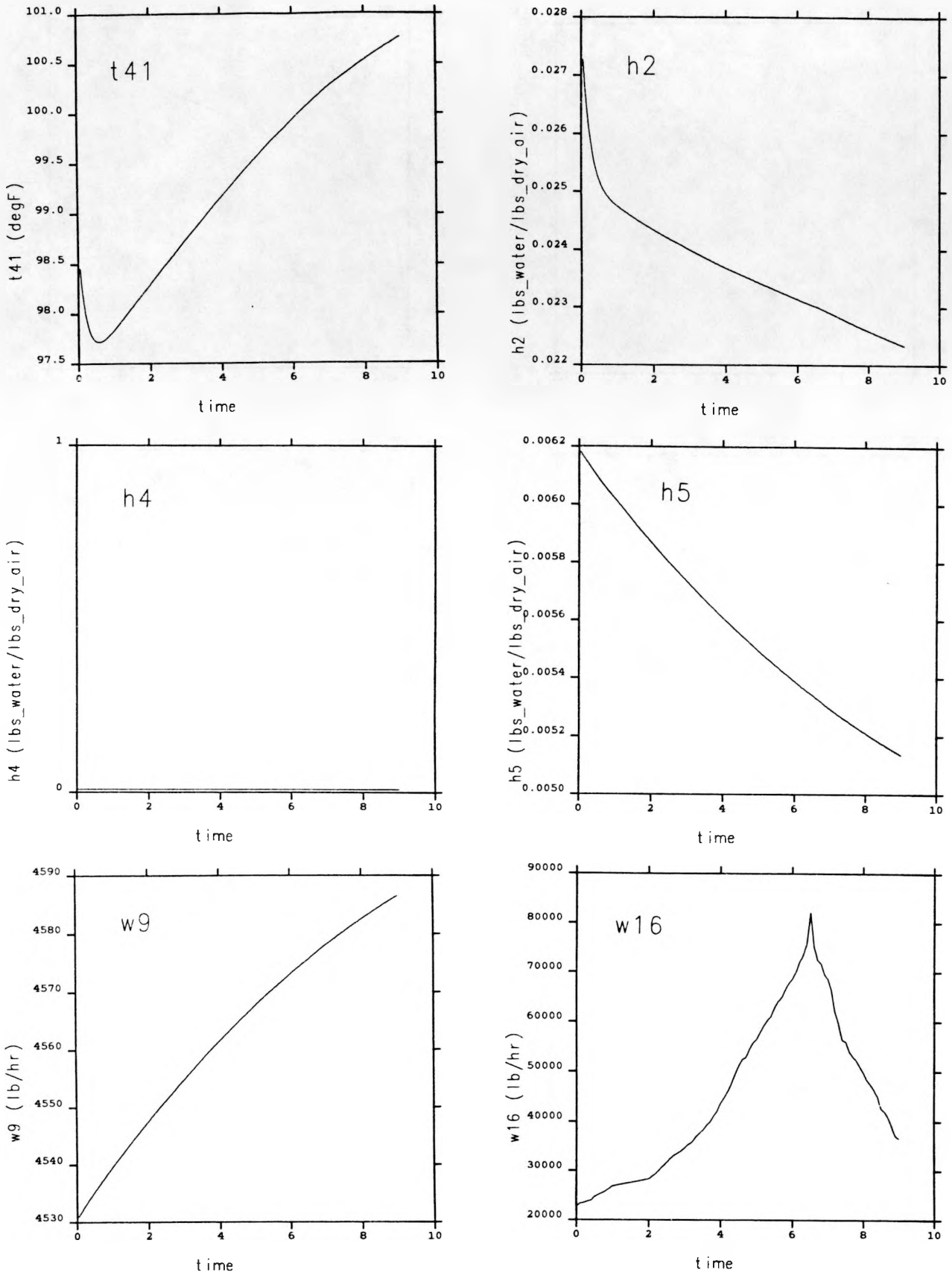


Figure 6 (cont.)

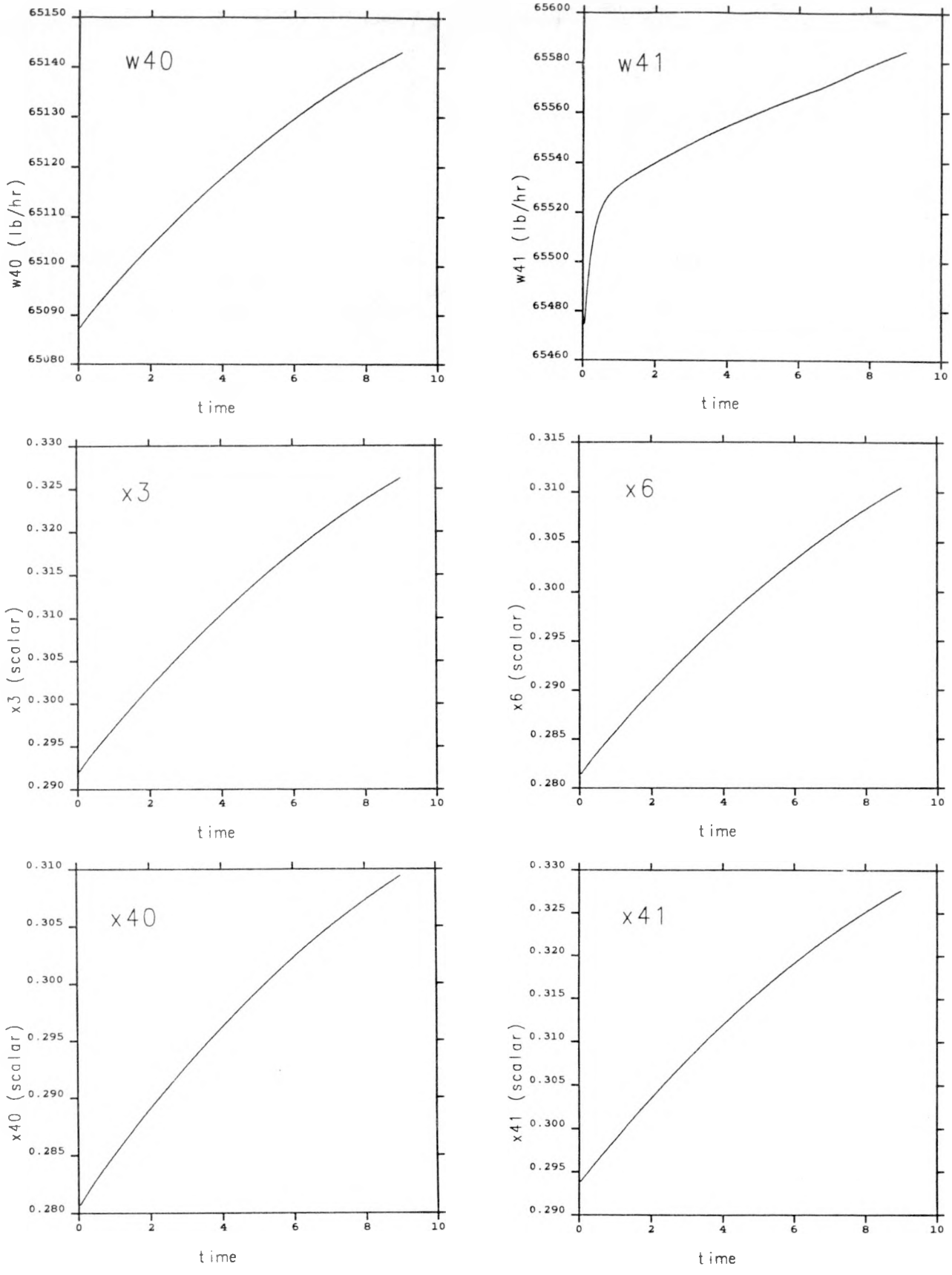
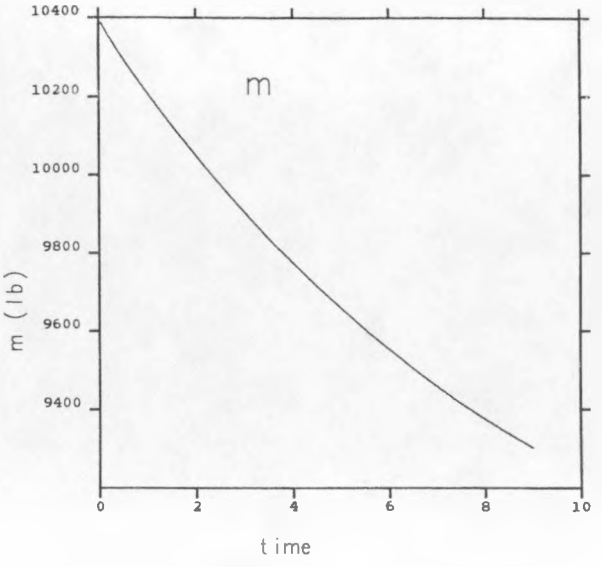
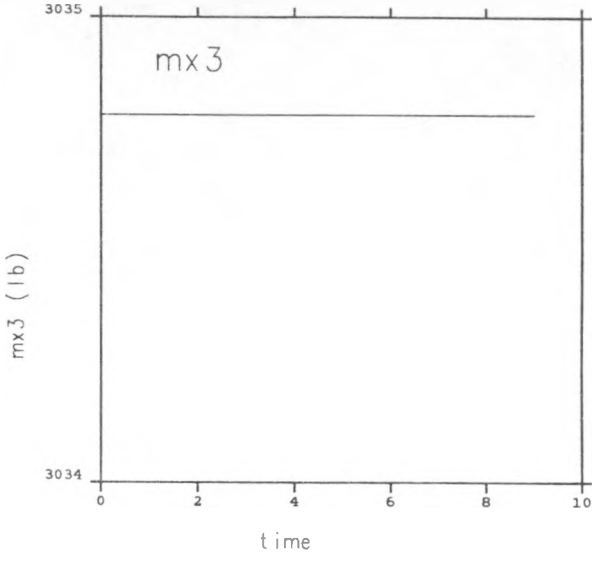


Figure 6 (cont.)


m
m (lb)
10400
10200
10000
9800
9600
9400
0
2
4
6
8
10
time


mx3
mx3 (lb)
3035
3034
0
2
4
6
8
10
time

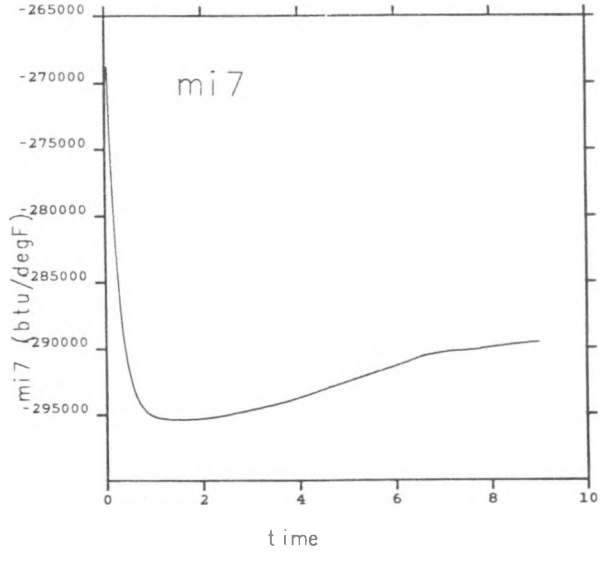

mi7
mi7 (btu/degF)
-265000
-270000
-275000
-280000
-285000
-290000
-295000
0
2
4
6
8
10
time

Figure 6 (end)