HETEROGENEOUS DECOMPOSITION AND INTER-LEVEL COUPLING FOR COMBINED MODELING

Paul A. Fishwick

Dept. of Computer & Information Science
University of Florida
Bldg. CSE, Room 301
Gainesville, FL 32611

ABSTRACT

Combined methods for modeling in simulation involve an integration of different model types. Often, discrete event methods are combined with continuous modeling techniques to model a complex environment that does not yield to formal description with only one model type. It is essential that, when defining a method for combined modeling, the coupling and decomposition methods are well defined. We demonstrate with an example model of boiling water that combined modeling is just a special case of using a heterogeneous approach to model composition. Using the heterogeneous approach provides a rich modeling environment where highly complex systems containing many distinct phases and multiple models can be conveniently represented.

1 INTRODUCTION

In a recent paper, Forbus (1988) presents a comprehensive overview of the field of qualitative physics. Forbus notes that "some device ontologies are unnatural" in modelling, for instance, a pot of boiling water (see fig. 1) or a bouncing ball when expressed within system dynamics. Although system dynamics (Shearer, Murphy and Richardson 1967) is inadequate for representing systems containing phase transitions and complicated boundary conditions. simulation methodology has developed concepts to model complex systems over multiple levels of abstraction (Fishwick 1986; Fishwick 1988; Fishwick 1989). Ören (1991) has developed a concept of *multimodel* to formalize models containing several submodels, only one of which is put into effect at any time. Cellier (1979) developed an approach to combined continuous/discrete event models implemented in a GASP language extension. Praehofer (1991) extended the Discrete Event System Specification (DEVS) (Zeigler 1990) to provide a formalism and a simulation environment for specifying combined continuous/discrete event models. In this article, we build on these developments by providing a methodology and formalism for developing multiple, cooperative models of physical systems of the type studied in qualitative physics. The formalism should help to build intelligent machines capable of physical modelling and reasoning.

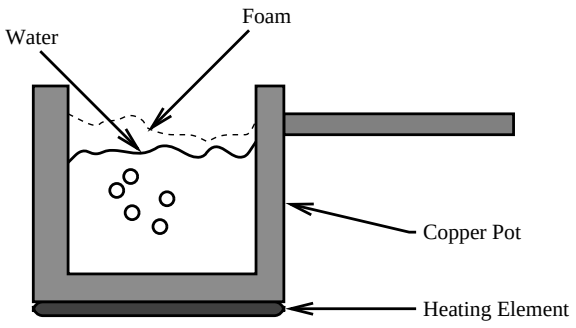

Figure 1: A Pot of Boiling Water

We will use the system of boiling water to illustrate our methods. Although at first glance, it appears too simplistic, the boiling water system is appropriate for demonstrating a wide range of discrete and continuous behaviors as well as levels of abstraction. All models for computer simulation are constructed to answer a certain class of question. With our multi-level approach, we are capable of answering a larger number of questions than with a single-level model. For instance, the question "How long will it take for the pot to boil over?" requires a numerical answer whereas "What is the next step after water starts heating?" involves a qualitative answer such as "If the system is in the phase *heating*, and the control knob is turned off then the next phase will be *cooling*. However, if the water temperature reaches 100 then the water starts *boiling*." We present a method that permits this kind of multi-level reasoning. Fishwick and Zeigler (1991) have recently discussed a method for linking the heterogeneous level coupling concept

within the formal DEVS framework.

We first review formal definitions of systems and key system concepts that are essential at any abstraction level. Then using the boiling water system as illustration, we link a finite state automaton (FSA) to an underlying continuous model thereby achieving a heterogenous level coupling.

2 SYSTEM DEFINITION

2.1 Formal Specification

A deterministic system $< T, U, Y, Q, \Omega, \delta, \lambda >$ within classical systems theory (Padulo and Arbib 1974; Wymore 1977) is defined as follows:

- T - is the *time* set. For continuous systems $T = \mathcal{R}$ (reals), and for discrete time systems, $T = \mathcal{Z}$ (integers).
- U - is the *input* set containing the possible values of the input to the system.
- Y - is the *output* set.
- S - is the *state* set.
- Ω - is the set of *admissible* (or *acceptable*) input functions. This contains a set of input functions that could arise during system operation. Often, due to physical limitations, Ω is a subset of the set of all possible input functions ($T \rightarrow U$).
- δ - is the transition function. It is defined as: $\delta : S \times T \times T \times \Omega \rightarrow S$.
- λ is the output function, $\lambda : T \times S \rightarrow Y$.

A system is said to be *time-invariant* when its behavior does not depend on the absolute value of time. In this case, the transition function can be simplified to the form: $\delta : S \times T \times \Omega \rightarrow S$. Here, $\delta(s, t, \omega)$ yields the state that results from starting the system in s and applying the input ω for a **duration** of t time units. In discrete-event like systems, the effect of the input can be captured in a resetting of the initial state.

The system formalism is very general. For instance, figures 2 and 3 show two sample models that can be represented using a 1) block network, and 2) state transition method, respectively. Both of these formalisms can be shown to be sub-classes of the system formalism (Zeigler 1976). In fig. 2, transfer functions are represented as boxes with inputs and outputs connected to each box. This is termed a "block model" in systems engineering or a "data flow" graph in software engineering. The state vector for this system

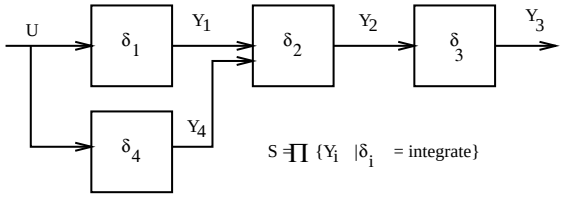

Figure 2: Block Network Orientation

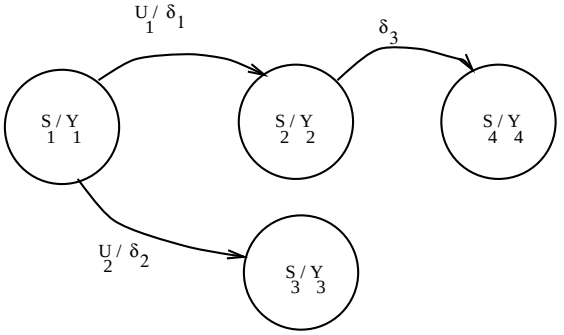

Figure 3: State Transition Orientation

is a subset of the outputs Y_i. Specifically, the outputs connected to boxes with memory (i.e., integrator or delay boxes) make up the system state ($\prod$ represents algebraic cross product):

$$S = \prod \{Y_i | \delta_i \equiv \int_0^t dt\}$$

Fig. 3 displays a system by emphasizing state-to-state transitions instead of using a block network orientation. An important class of state transition model is the finite state automaton. In this example, outputs Y_i are associated with states S_i. Input conditions involving U_i provide determinism where there are multiple branches from a state.

3 COMBINED MODELS

Taking the cross product of time and state in terms of two possible values ("discrete" and "continuous") suggests four possible model types. The *Discrete Event* model type has continuous time and a discretized state space. A *Discrete Time* model has a discrete time space with equal time intervals (the state space may be either discrete or continuous). A *Continuous* model has both continuous and discrete time and space. Table 1 displays these combinations with example model formalisms for each. What about *continuous* events? In the simulation literature (Oren 1987a; Oren 1987b), one finds reference only to discrete events. Continuous events might be defined in

terms of the start and end of an arbitrary numerical integration interval. However, this concept is not adequate since it depends on a simulation process, and is not an intrinsic characteristic of the model. It seems that events, by their very nature, are discrete since they map to cognitive and linguistic concepts connected with the processes that we model. In the next section, we attempt to provide a conceptual framework for understanding discrete events.

A *combined* model combines two or more of the above model types. For instance, a combined discrete event/continuous model has two distinct model types: a discrete event model and a continuous model. These two models are coupled with discrete events (Cellier 1979; Praehofer 1991).

4 BOILING WATER EXAMPLE

4.1 A High Level Automaton Model

Consider a pot of boiling water on a stovetop electric heating element. Initially, the pot is filled to some predetermined level with water. A small amount of detergent is added to simulate the foaming activity that occurs naturally when boiling certain foods. This system has one input or control – the temperature knob. the knob is considered to be in one of two states: on or off (on is $190°C$; off is α - ambient temperature). We make the following assumptions in connection with this physical system:

1. The input (knob turning) can change at any time. The input trajectories are piecewise continuous with two possible values (ON,OFF).

2. The liquid level (height) does not increase until the liquid starts to boil.

3. When the liquid starts to boil, a layer of foam increases in height until it either overflows the pot or the knob is turned off.

4. The liquid level decreases during the heating and overflow phases only.

To create a mathematical model, we must start with data and expert knowledge about the domain. If enough data can be gathered in a cost effective way then our model engineering process will be simplified since we will not have to rely solely on heuristics to identify the model. By analyzing a pot of boiling water we may derive simple causal models whose individual transitions may be $knob\_on \Rightarrow water\_getting\_hotter(1.0)$ or $water\_getting\_hotter \Rightarrow water\_boiling(0.75)$ where numbers in parentheses are certainty factors. An important facet of system modelling is that we choose certain modelling methods that require a categorization of informally specified system components. Key components of any system model are *input*, *output*, *state*, *event*, *time* and *parameter*. Different modelling methods include these components in different ways. For instance, an FSA focuses on state-to-state transitions with input being labeled on each arc. A dataflow model, on the other hand, focuses on the transfer function between input and output. We define our most abstract model of boiling water using an FSA shown in figure 4.

4.2 Heterogeneous Model Refinement

Homogeneous model refinement (Fishwick and Zeigler 1991) is the process of refining models of the same type. For instance, we might represent the boiling water system using a hierarchy of finite state automata instead of the single level shown in fig. 4. We will limit our discussion in this paper to heterogeneous refinement. Heterogeneous refinement takes homogeneous refinement a step further by loosening the restriction of equivalent model types. For instance, we might have a Petri net at the high abstraction level and we may choose to decompose each transition into a block graph so that when a transition fires within the Petri net, one may "drop down" into a functional block level. For the FSA in fig. 4 we choose to represent each state as a continuous model. Specifically, each state will define how three state variables, T (temperature), H_w (height of water), and H_f (height of foam on the top of the water) are updated. In all cases, $H_f \geq H_w$. The end result will eventually be a multimodel that will be coordinated by the FSA.

4.3 A Low Level Continuous Model

The continuous models contained within states $Heating$ and $Cooling$ (shown in fig. 4) require some physical theory before stating them. We model heat conduction since convection and radiation do not play major roles in this system. To derive a good continuous model for $Heating$ and *cooling*, we first define resistance. There is thermal resistance $R = H/kA$ (H is the height of water, A is the surface area of the pot, and k is the thermal conductivity of water). We will ignore (or "abstract out") the resistance of the pot since it is not as significant as the resistance of water. The definition for thermal capacitance C is $C\dot{T} = q_h$ with q_h being the flow of heat from the heating element to the water. We will let C_1 be the capacitance of the metal pot, C_2 be the capacitance of water, and C be the total capacitance. Newton's law of cooling states that $Rq_h = \Delta T = T_1 - T_2$ where

Table 1: Simulation Model Types

	Discrete Space	Continuous Space
Discrete Time	*Discrete Time*	*Discrete Time*
	Difference Equations (with integer states) Cellular Automata Finite State Automata	Difference Equations (with real states)
Continuous Time	*Discrete Event*	*Continuous*
	Queuing Models Digital Logic Models	Differential Equations

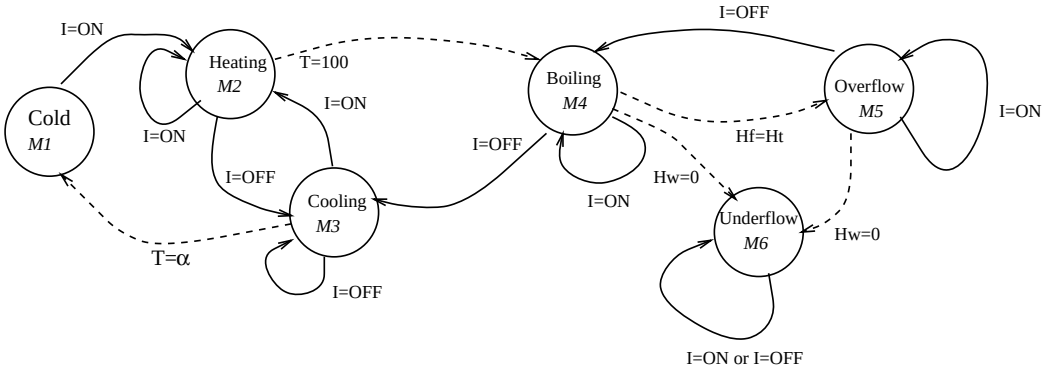

Figure 4: Six State Automaton Controller for Boiling Water Multimodel

T_1 is the temperature of the source (heating element), and T_2 is the temperature of the water. Since T_2 is our state variable we let $T = T_2$ for convenience. By combining Newton's law with the capacitance law, and using the law of capacitors in series, we arrive at:

$$k = \frac{C_1 + C_2}{RC_1C_2} \quad (1)$$

$$\dot{T} = k(T_1 - T) \quad (2)$$

Hence, equation (2) is a first order lag with a step input representing the sudden change in temperature as effected by a control knob. Figure 5 displays a block diagram of *Heating* within the *Heating* state. Proper coupling is essential in heterogeneous refinements. That is, it must be made clear how components at one level match components at the higher level. Note, in fig. 5, the transfer function taking the ON/OFF input detected by the FSA and converting these input values to temperature values for the block network. Specifically, the block labeled 'F' performs the mapping from 'ON/OFF' to real-valued temperatures $\alpha \leq T \leq 190$.

The low-level continuous models $M_1, \ldots, M_5$ are defined as follows:

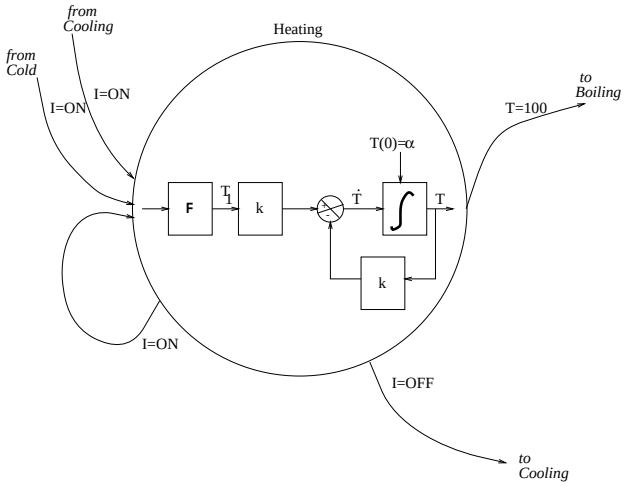

Figure 5: Decomposition of *Heating* State

1. (M_1) COLD: $T = \alpha$, $\dot{H_w} = 0$, $\dot{H_f} = 0$.
2. (M_2) HEATING: $\dot{T} = k_1(100 - T)$, $\dot{H_w} = 0$, $\dot{H_f} = 0$.
3. (M_3) COOLING: $\dot{T} = k_2(\alpha - T)$, $\dot{H_w} = 0$, $\dot{H_f} = -k_3(H_f - H_w)$.
4. (M_4) BOILING: $T = 100$, $\dot{H_w} = -k_4 H_w$, $\dot{H_f} = k_5(H_f - H_w)$.
5. (M_5) OVERFLOW: *same as BOILING* w/ constraint $H_f \geq H_t$.
6. (M_6) UNDERFLOW: $T = undefined$, $H_w = H_f = 0$.

The system phase is denoted by Φ and the state variables are:

- T: temperature of water.
- H_w: height of the water.
- H_f: height of the foam.

Note that the continuous models share a common set of state variables. However, in general state variables may be different for each M_i model.

There are also some constants such as H_t for the height of top of pot, H_s for the starting height of water when poured into the pot; and k_i rate constants. The initial conditions are: $\Phi = cold$, $T(0) = \alpha$, $H_w(0) = H_f(0) = H_s$ and $knob = OFF$. By including the functional block knowledge, we create one large model called COMBINED that is defined as the FSA in fig. 4 with each state containing a block model (as shown in fig. 5).

5 FORMING A MULTIMODEL

5.1 General Case

We have presented a refinement of graphs from the FSA in fig. 4 the block graph at the lowest level. To form a multimodel, we need to specify how the set of models that we have defined can be coordinated so that exactly one submodel is active at any time.

A multimodel can be created from fig. 4 by treating each of the states as a phase, and associating a model with each phase. For ease of exposition, before considering the general case, let's consider an example. Figure 6 displays a phase graph with 3 phases A, B, and C.

A different model is associated with each phase; for instance, in phase B, the underlying model is M_1 will generate the state trajectories. External events are denoted in fig. 6 with a solid arc. If in phase

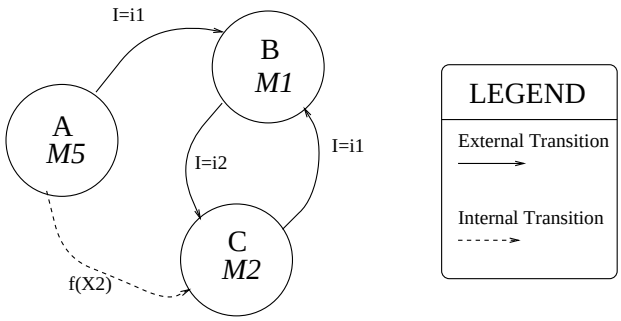

Figure 6: Generic Phase Graph

A, for instance, and an input of i_1 is received, the phase immediately moves to B and the model is M_1 starting in a state determined by the state of M_5 when the input i_1 occurred; otherwise the phase remains A. Internal (also called state) events occur when the state of a model satisfies specific transition conditions. They are denoted by dashed lines from one phase to another. For instance, in phase A, let $f(X_2) \equiv equal(X_2, 4.2)$. This is interpreted as follows: if the state variable X_2 "reaches" the value 4.2 (i.e., $X_2 = 4.2$) then the phase becomes C.

Note that a phase graph is an FSA augmented with the capability of recognizing changes in state events (internal events) as well as input events (external events). The phase graph will become the means of coordinating the transitions and activations of the submodels in a multimodel.

Let us summarize our example formulation of an *FSA-controlled multimodel*. In fig. 6, the following variables are used:

- A, B, and C are phases.
- I is an input variable. i_1 and i_2 are the values that I can assume; the input has an immediate effect on the phase.
- X_2 is a sample state variable in the continuous model M_5.
- f is an arbitrary predicate with a true or false value.

Such an FSA-controlled multimodel can be simulated in a combined continuous/discrete event environment such as those of Cellier (1979) and Praehofer (1991). However, to fully understand the operation of such simulation, we first present a formalization.

5.2 Heterogeneous Level Coupling

Using the method described in the previous section, we define an *FSA-controlled multimodel* of the boil-

ing water system by using the automaton in fig. 6 to produce the FSA in fig. 4. The FSA controls the simulation; however, it is also a complete system description by itself. Each phase within the FSA controls a specific block model ($M_1, \ldots, M_5$). Now we demonstrate the relationship between the FSA and each block model according to our system theoretic description in section 2.1. First, we require a definition of partition:

Definition: *Partition for Boiling Water System.* Let $S = \mathcal{R}^3$. Let π be a partition of S defined as:

$$\pi = \{P_1, P_2, P_3, P_4, P_5\}$$

where:

1. $\forall i \in \{1,2,3,4,5\}, P_i \subseteq S \wedge P_i \neq \phi$.
2. $\forall i P_i$ form a cover for S s.t. $S = P_1 \cup \ldots \cup P_5$.
3. $\forall i P_i$ are mutually disjoint ($i \neq j \Rightarrow P_i \cap P_j = \phi$.

The FSA in fig. 4 is an abstraction of $M_1, \ldots, M_5$ in the following way. $S = \mathcal{R}^3$ is the state space for all M_i. Recall that the state for an M_i is defined by a triple (T, H_t, H_f). We partition the state space S and assign a name to each partition (i.e. phase): heating, cooling, boiling, underflow, overflow. Given that systems can be formally defined as $< S, I, O, \delta, \lambda >$, we define fig. 4 as follows:

- $S' = \{P_1, \ldots, P_5\}$
- $I' = \{ON, OFF\}$
- $O' = S'$
- $\delta' : S' \times I' \rightarrow S'$
- $\lambda' : S' \rightarrow O'$

whereas each M_i is defined as follows:

- $S = \mathcal{R}^3$
- $I = \mathcal{R}$
- $O = S$
- $\delta : S \times I \rightarrow S$
- $\lambda : S \rightarrow O$

There, now, is a definite relationship between the two types of systems: S' represents a set of partitions of S; I' relates to I using the map F depicted in fig. 5: $F : I \rightarrow I'$ and δ' and δ represents a local transition function and block flow models, respectively, that define state-to-state transitions. The relationships between the partition of S and the geometry of phase space is depicted in figures 7 and 8. In figs. 7 and 8, phase names point to the region of phase space to which they refer. In fig. 7, the phase *Underflow* refers to the plane where $H_w = 0$. The other 3 phases are rectangular polyhedra. In fig. 8, phase *Cold* is the plane where $T = \alpha$, and *Cooling* is the entire rectangular block minus the other two phases. Note that the phase space partitioning is dependent on the current value of the input ($I = OFF$ or ON).

The two systems S' and S are formally related to one another using the system homomorphism (Zeigler 1976). We create three homomorphic mappings h_1,h_2 and h_3:

1. $h_1 : S \rightarrow S'$.
2. $h_2 : I \rightarrow I'$.
3. $h_3 : O \rightarrow O'$.
4. $\forall s \in S, i \in I(\delta'(h_1(s), h_2(i)) = h_1(\delta(s, i)))$.
5. $\forall s \in S(\lambda'(h_1(s)) = h_3(\lambda(s)))$.

Note that h_2 is equivalent to the F function in fig. 5. A table defining the relationships between the state variables of S (T,H_w,H_f) and the state and input variables of S' is depicted in table 2. The symbol ϕ means "undefined value."

6 MULTIMODEL SIMULATION

To describe the simulation of multimodels, we present algorithms that are implementable in *SimPack* (Fishwick 1990) (a set of simulation tools). An FSA-controlled multimodel can be converted to a discrete event simulation by translation of its graph to the core of a discrete event simulation: the event switch statement. The switch statement is composed of event statement blocks (or "routines") that gain control of the simulation when a scheduled event on a future event list is initiated (or "caused"). To show this, we use the following routines:

- *next_event(&E)* takes the next event from the front of the future event list and places it in E.
- *schedule(E, T)* schedules an event to occur by placing it in the future event list. E is the event that is to occur, and T is the amount of time to elapse before executing event E.
- *instantiate(M,Q)* provides a context switch so that continuous model M now "controls" the system behavior. *instantiate* causes a new initial state Q for M so that simulation can begin. Each

Table 2: Relationship Between Variables of S and S'

S	I	T	H_w	H_f
COLD	ON	ϕ	ϕ	ϕ
COLD	OFF	$= \alpha$	$> 0 \wedge < H_t$	$> 0 \wedge < H_t$
HEATING	ON	$> \alpha \wedge < 100$	$> 0 \wedge < H_t$	$> 0 \wedge < H_t$
HEATING	OFF	ϕ	ϕ	ϕ
COOLING	ON	ϕ	ϕ	ϕ
COOLING	OFF	$> \alpha \wedge < 100$	$> 0 \wedge < H_t$	$> 0 \wedge < H_t$
BOILING	ON	≥ 100	$> 0 \wedge < H_t$	$> 0 \wedge < H_t$
BOILING	OFF	≥ 100	$> 0 \wedge < H_t$	$> 0 \wedge < H_t$
OVERFLOW	ON	≥ 100	$> 0 \wedge < H_t$	$\geq H_t$
OVERFLOW	OFF	ϕ	ϕ	ϕ
UNDERFLOW	ON	ϕ	$= 0$	$= 0$
UNDERFLOW	OFF	ϕ	$= 0$	$= 0$

M has a set of state variables and state variable relationships (such as a set of differential equations).

- *time_state(C,M)* yields the time elapsed in model M for condition C to become true.
- *save-state()* saves the state of the model in the current phase.
- *get-state()* gets the current state saved previously by *save-state()*.
- *time_input(I)* yields the time elapsed until a specific input enters the system.

This scheme presumes a deterministic simulation where the input stream to be fed into the simulation is produced -and therefore known- in advance. In this way, it is not difficult to determine which event comes first: (1) a specific input value, or (2) a state event occurrence.

Below, we present the switch statement corresponding to the graph in fig. 6:

Algorithm 1

```
next_event(&event);
save-state();
switch(event) {
/* external events - list all input values as cases */
i1:     switch(phase) {
                A: schedule(B,0); break;
                C: schedule(B,0); break;
        } /* end switch */
        break;
i2:     if(phase == B) schedule(C,0); break;
/* internal events - list all phases in graph as cases */
A:      phase = A;
        instantiate(M5,get-state());
        if (time_state(f(X2),M5) < time_input(i1))
           schedule(C,time_state(f(X2),M5));
        break;
B:      phase = B;
        instantiate(M1,get-state()) break;
C:      phase = C;
        instantiate(M2,get-state()) break;
} /* end switch */
```

The event routine algorithm correponding to fig. 4 is:

Algorithm 2

```
next_event(&event);
save-state();
switch(event) {
/* external events - list all input values as cases */
on:     switch(phase) {
                cold: schedule(heating,0); break;
                cooling: schedule(heating,0); break;
        } /* end switch */
        break;
off:    switch(phase) {
                heating: schedule(cooling,0); break;
                boiling: schedule(cooling,0); break;
                overflow: schedule(boiling,0); break;
        } /* end switch */
        break;
/* internal events - list all phases in graph as cases */
cold:   phase = cold;
        instantiate(M1,get-state()); break;
heating: phase = heating;
        instantiate(M2,get-state());
```

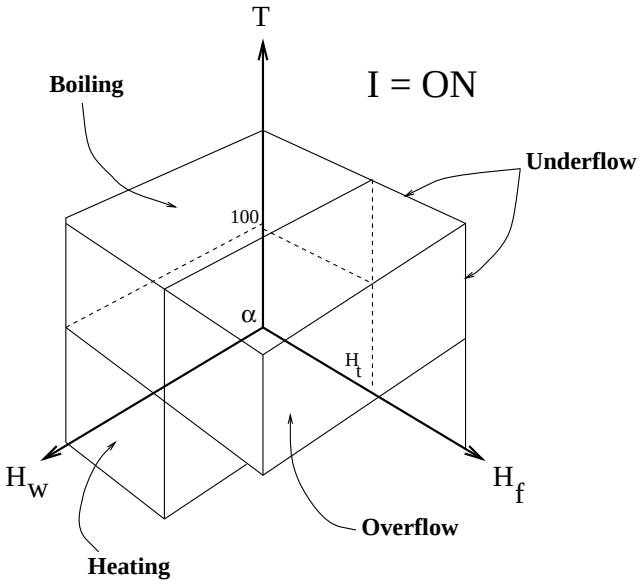

Figure 7: Phase Space Partitioning When I=ON

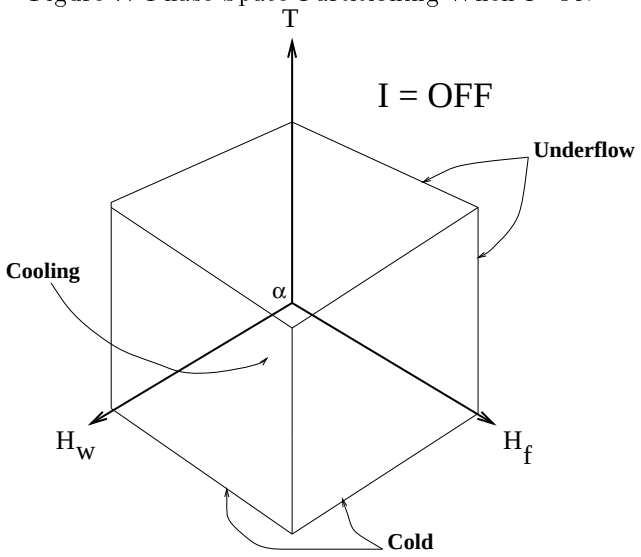

Figure 8: Phase Space Partitioning When I=OFF

```
                if(time_state(equal(T,100),M2)
                  < time_input(off))
                  schedule(boiling,time_state(equal(T,100),
                    M2));
                break;
cooling:        phase = cooling;
                instantiate(M3,get-state());
                if(time_state(equal(T,α),M3)
                  < time_input(on))
                  schedule(cold,time_state(equal(T,α),
                    M3));
                break;
boiling:        phase = boiling;
                instantiate(M4,get-state());
                if(time_state(equal(Hf,Ht),M4)
                  < time_input(off))
                  schedule(overflow,time_state(equal(Hf,Ht),
                    M4));
                break;
overflow:       phase = overflow;
                instantiate(M5,get-state());
                if(time_state(equal(Hw,0),M5)
                  < time_input(off))
                  schedule(underflow,time_state(equal(Hw,0),
                    M5));
                break;
underflow:phase = underflow;
```

We remark that a more careful statement of some of the conditions is actually required in the above. This occurs when the state trajectory in a model approaches, but never actually reaches, the threshold specified in the condition. For example, the heating model exponentially approaches the boiling temperature. In such cases, a tolerance around the limiting value has to be set which will provide a definite boundary crossing.

7 Conclusions

We have used a system of boiling water to demonstrate how to create an FSA-controlled multimodel which efficiently represents a system using heterogeneous models. The models represent a "combined" simulation in that there is an automaton level and a functional block level. A combined continuous/discrete-event model divided up the total boiling water process into distinct phases. Coordination of models for these phases was facilitated by the phase graph associated with the FSA. In addition, we demonstrated the formal system theoretic relationships between the FSA and block model definitions. We found that by geometrically partitioning

phase space we could form an FSA with equivalent behavior to the collection of individual differential equation based systems representing the low level state changes.

ACKNOWLEDGMENTS

We would like to thank Bernard Zeigler and Herbert Praehofer for detailed discussions on the topic of combined modeling.

REFERENCES

Cellier, F. E. 1979. *Combined Continuous System Simulation by Use of Digital Computers: Techniques and Tools.* PhD thesis, Swiss Federal Institute of Technology Zurich.

Fishwick, P. A. 1986. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction.* PhD thesis, University of Pennsylvania.

Fishwick, P. A. 1988. The Role of Process Abstraction in Simulation. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):18 – 39.

Fishwick, P. A. 1989. Abstraction Level Traversal in Hierarchical Modeling. In Zeigler, B. P., Elzas, M., and Oren, T., editors, *Modelling and Simulation Methodology: Knowledge Systems Paradigms*, pages 393 – 429. Elsevier North Holland.

Fishwick, P. A. 1990. Computer Simulation Modeling and Analysis: Methodology and Algorithms. unpublished simulation course notes.

Fishwick, P. A. and Zeigler, B. P. 1991. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation.*

Forbus, K. 1988. Qualitative Physics: Past, Present and Future. In Shrobe, H., editor, *Exploring Artificial Intelligence*, pages 239 –296. Morgan Kaufmann.

Oren, T. I. 1987a. Simulation Model Symbolic Processing: Taxonomy. In *Systems and Control Encyclopedia*, pages 4377 – 4381. Pergammon Press.

Oren, T. I. 1987b. Simulation: Taxonomy. In *Systems and Control Encyclopedia*, pages 4411 – 4414. Pergammon Press.

Oren, T. I. 1991. Dynamic Templates and Semantic Rules for Simulation Advisors and Certifiers. In *Knowledge Based Simulation: Methodology and Application*, pages 53 – 76. Springer Verlag.

Padulo, L. and Arbib, M. A. 1974. *Systems Theory: A Unified State Space Approach to Continuous and Discrete Systems.* W. B. Saunders, Philadelphia, PA.

Praehofer, H. 1991. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation.* PhD thesis, Johannes Kepler University of Linz.

Shearer, J. L., Murphy, A. T., and Richardson, H. H. 1967. *Introduction to System Dynamics.* Addison Wesley.

Wymore, A. W. 1977. *A Mathematical Theory of Systems Engineering: The Elements.* Krieger Publishing Co.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation.* John Wiley and Sons.

Zeigler, B. P. 1984. *Multi-Facetted Modelling and Discrete Event Simulation.* Academic Press.

Zeigler, B. P. 1990. *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems.* Academic Press.

AUTHOR BIOGRAPHY

PAUL A. FISHWICK is an associate professor in the Department of Computer and Information Sciences at the University of Florida. He received the BS in Mathematics from the Pennsylvania State University, MS in Applied Science from the College of William and Mary, and PhD in Computer and Information Science from the University of Pennsylvania in 1986. He has experience at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a member of IEEE, IEEE Society for Systems, Man and Cybernetics, IEEE Computer Society, The Society for Computer Simulation, ACM and AAAI. Dr. Fishwick was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, *The Transactions of the Society for Computer Simulation*, *International Journal of Computer Simulation*, and the *Journal of Systems Engineering.*