Multimodel Control of Continuous Processes through State Space Phase Partitioning*

Paul A. Fishwick
University of Florida

Abstract

The dynamics of systems are qualitatively characterized by phase transitions over system state spaces. We develop a methodology where a state space is partitioned and then partitions are lumped into phases which are combined to form finite state automata controllers of the underlying continuous system. We demonstrate the use of partitioning to create multimodels delineating levels of abstraction, and we further demonstrate that phases can refer not only to semantics for a single model, but also, to semantics for different models — one for each phase. The multimodel approach provides a method of combining different parts of a system together, permitting a wide variety of questions to be asked of the system during simulation.

1 Introduction

An ongoing problem within simulation modeling is how to manage multi-component models. The simulation of complex systems requires that we construct hierarchical models with multiple levels of abstraction [3, 4, 5]. While it is not always straightforward to define precisely how each level is constructed, we have found a class of automata driven models to be expressive in modeling complex systems that have both continuous and discrete event components. We term this class of models *multimodels* [6, 7]. Ören [9] developed a concept of multimodel to formalize models containing several submodels, only one of which is put into effect at any given time. The idea of a multimodel has its roots within the work in *combined* simulation modeling which has traditionally referred to an integration of discrete event and continuous modeling within the same system description. Pritsker [15, 16] first implemented combined modeling in the GASP modeling language, while Cellier [1] developed an approach to combined continuous/discrete event models implemented in a GASP language extension. Praehofer [12] extended the Discrete Event System Specification (DEVS) [19] to provide a formalism and a simulation environment for specifying combined continuous/discrete event models.

*Author's Address: Paul A. Fishwick, Dept. of Computer and Information Sciences, University of Florida, Bldg. CSE, Room 301, Gainesville, FL 32611. This research was partially supported by a grant from the Florida High Technology and Industrial Council (FHTIC) entitled "Real Time Computer Animation for Interactive Visualization."

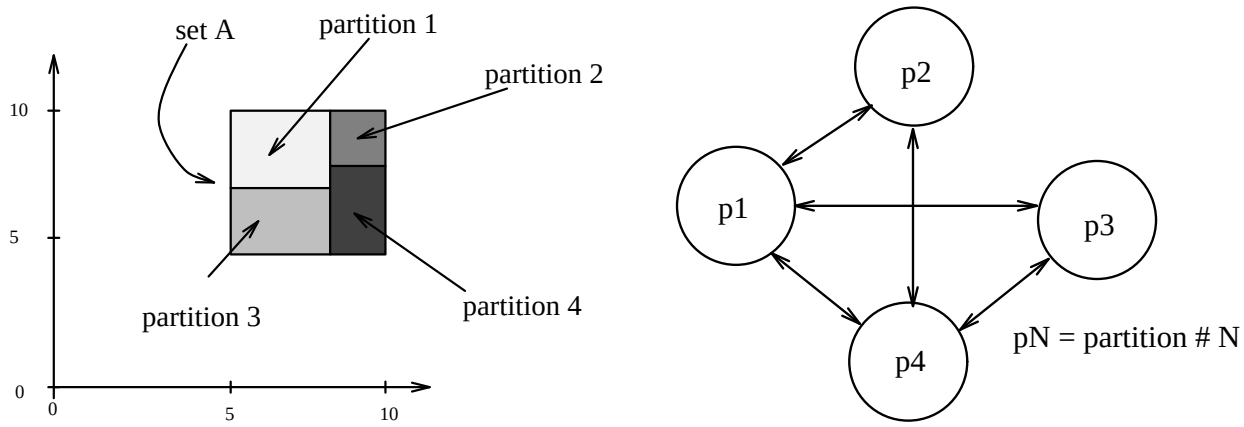
Our approach will be to first discuss the generic concepts of phase space [11] and partitions while noting their importance to dynamics in specific and to multimodeling in general. We then provide examples of multimodeling and its relationship to phase partitioning. A formal specification for the automaton-driven multimodel can be expressed in slightly simplified version of the DEVS formalism [18] which we term $DEVS'$. We close the discussion with future research directions and conclusions of this work.

2 Multimodeling

2.1 Method of Partitioning

Prigogine and Nicolis [14, 8] stress the concept of "symmetry breaking" in complex systems where symmetry breaking can be defined by the qualitative changes to system behavior after a bifurcation occurs in phase space. When a system has a relatively simple pattern of behavior, there are a small number of attractors and basins of attraction constituting phase space. The small number of basins characterizes the system as being symmetric (or homogeneous) with respect to operations that effect the system. However, as certain parameters or external inputs are modified, systems become more complex by exhibiting a variety of distinct qualitative behaviors. Eventually, a system undergoing symmetry breaking, will continue to evolve into chaos where sensitivity to initial conditions and parameter changes predominate. Although, bifurcations are known to be at the heart of many nonlinear systems, the partitioning of phase space, in general, is central to the way we can model *all* systems. We begin with a short discussion of partitioning and system morphisms.

Let A and B be sets. A is said to be a *subset* of B ($A \subset B$) if each element of A is an element of B. Therefore, if $A = \{0, 2\}$ and $B = \{0, 1, 2, 3\}$ then $A \subset B$ since $0 \in A \Rightarrow 0 \in B$ and $2 \in A \Rightarrow 2 \in B$. The definition of *partition* uses the subset concept. A partition π of a nonempty set A is a collection of non-empty subsets of A such that: (1) for all $X \in \pi$ and $Y \in \pi$, either $X = Y$ or $X \cap Y = \phi$, and (2) $A = \bigcup_{S \in \pi} S$. Partitions of sets are best viewed in pictorial form as in figure 1(a) where we have set A which is a two-dimensional subset of $\mathcal{R}^2$ where $\mathcal{R}$ is the set of real numbers. A is partitioned into four subsets. Depending on the connectivity in phase space, we can create a finite state automaton (FSA) to control motion through the phase space using a forcing function. An automaton reflecting control through phase space is shown in figure 1(b). Mappings between two sets are shown using functions. For instance, the state transition function, δ for some model M takes a state and input and determines a state transition. If a model has a state space $S = \{s_1, s_2\}$ and a transition function such that $\delta(s_1, 0) = s_2$, $\delta(s_1, 1) = s_1$ and $\delta(s_2, 0) = s_1$ then we can form a map that visually defines δ:



(a) Partition of set A.

(b) FSA for partition in (a).

Figure 1: Partitioning phase space.

$$(s_1, 0) \longrightarrow s_2$$

$$(s_1, 1) \longrightarrow s_1$$

$$(s_2, 0) \nearrow$$

We define model M_1 to be an abstraction of model M_2 if one can form homomorphic mappings between each respective model component (input, output and states). Moreover, the mappings must preserve the existing functional behavior of δ and λ. Let S be the state set for M_2 and S' be the state set for M_1. The preservation of the transition functions for each system (δ_1 and δ_2) is shown below:

$$\begin{array}{ccc} S & \xrightarrow{\delta_1} & S \\ {\scriptstyle h_1}\downarrow & & \downarrow{\scriptstyle h_1} \\ S' & \xrightarrow[\delta_2]{} & S' \end{array}$$

To refine model M_1 (level one) into model M_2 (level two), we adhere to the following

two rules:

1. Convert each model to the canonical systems form defined by Padulo and Arbib [10] and extended by Zeigler [17].

2. Create homomorphic mappings between respective components of M_1 and M_2. First consider each member of a component (such as state) set. Map a partition of the state space at level two (the lower abstraction level) to an element of the state space for level one (the higher abstraction level).

By mapping partitions of M_2 to states of M_1 (an FSA by our example), we create a "controller" for the lower level process, M_2 that operates over a continuous phase space. Consider the dynamics of an undamped, unforced pendulum (i.e., damping factor of $\epsilon = 0.0$):

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -\epsilon x_2 - sin(x_1) \end{aligned}$$

Figure 2 displays the phase trajectories and phase region A (defined below). Figure 3 shows phase regions A and B. Since phase area is conserved during flow, the phase trajectories serve to describe equipotential energy contours. The regions display basins of attraction that partition the state space. The topology of the phase space provides us with three distinct regions (after considering the symmetry by "wrapping" the phase space onto a cylinder):

- (Region A) CCW-Rotation: The bob swings counter-clockwise in a complete orbit.
- (Region B) Vibration: The bob oscillates back and forth[1].
- (Region C) CW-Rotation: The bob swings clockwise in a complete orbit.

To control the pendulum requires a forcing function of the general form $f(t)$ so that it is possible to traverse from one basin to another[2]. Note that the geometry of the basins will change depending on parameters such as ϵ and the input trajectory $f(t)$; however, our partitioning of phase space is independent of these modifications since regions A,B and C are fixed and meaningful in any context regardless of basin geometry. We create an FSA with three states, one for each phase region in fig. 2(a). Note that for our pendulum example, there is one underlying equational model; however, we need not use the same mathematical model when transitioning from one phase to another. Later, we consider other examples that demonstrate that each phase can represent a different model[3].

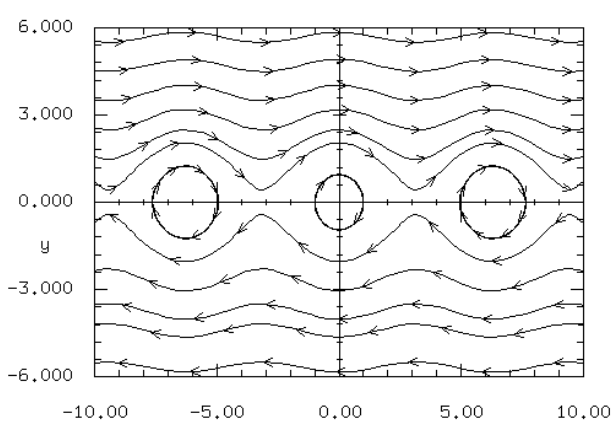
We control the pendulum by using $f(t)$ and the boundaries of regions A, B and C are equal to the separatrices present in the phase graph. These dividing lines provide a method where we can label the FSA appropriately. The Hamiltonian for the pendulum is

$$H(\theta, p) = \frac{p^2}{2m} + V(\theta)$$

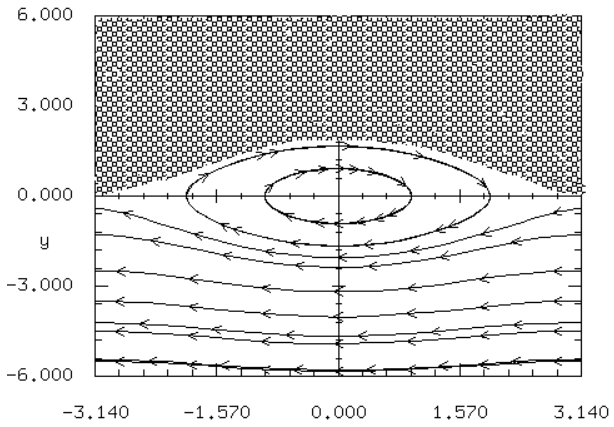
[1] Also known as *libration*.

[2] The general method for basin switching can be chosen from the literature on the control of nonlinear systems [2] which includes sub-optimal and optimal methods.

[3] Where each model may have different state spaces.

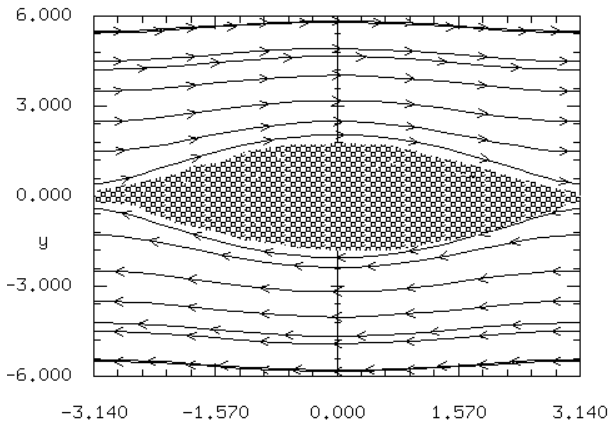


(a) Phase space.

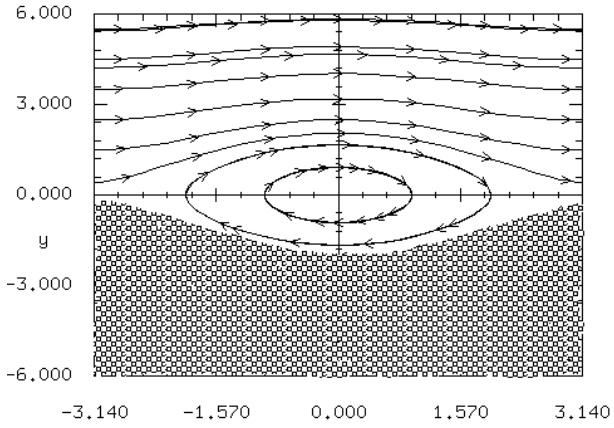


(b) Phase region A.

Figure 2: Phase space and region A.



(a) Phase region B.



(b) Phase region C.

Figure 3: Regions B and C.

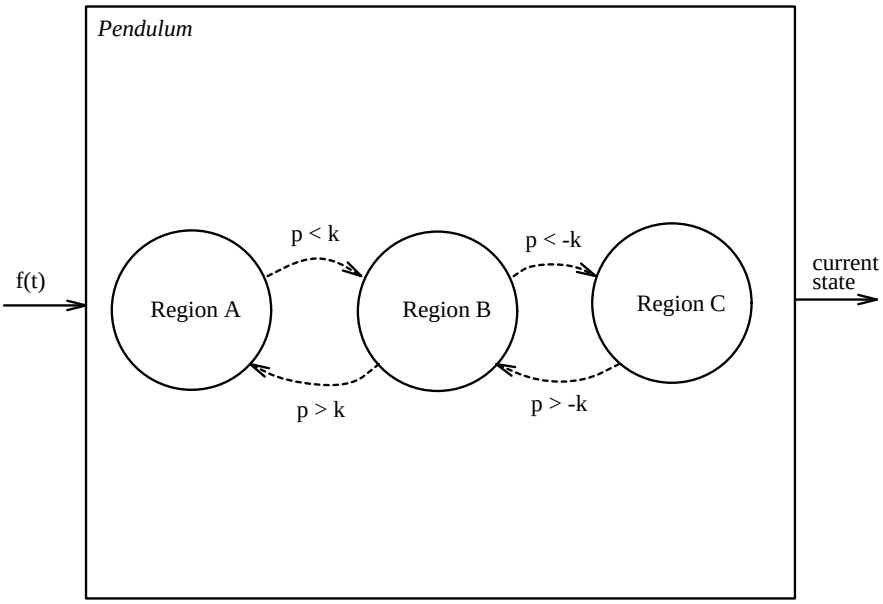


Figure 4: FSA pendulum control.

where $V(\theta) = -\alpha^2 cos(\theta)$ and $\alpha = (mgl)^{\frac{1}{2}}$. This provides a method for measuring the energy level for any (θ, p) in the phase plane. The separatrix defining the boundary between regions A, B and C is determined by first noting that the potential energy will be at its maximum when the pendulum bob at located at $\theta = \pi$. Given this, we obtain $V(\theta) = -\alpha^2$. Therefore, we solve the following equation

$$\alpha^2 = \alpha^2 cos(\theta) - \frac{1}{2}p^2$$

to determine the separatrix equation(s):

$$p = \pm 2\alpha cos(\frac{1}{2}\theta).$$

This provides the FSA in figure 4 where $k = 2\alpha cos(\frac{1}{2}\theta)$.

2.2 Refining Aggregate Vectors

Many simulations involve the motion of some number, n bodies in a d dimensional space. Partitioning space —for these kinds of systems— is, perhaps, the most straightforward type of state space partitioning. We will take an aggregate vector (which can be either a single number or vector of numbers) such as the center of mass for n bodies and then partition a lower level state space accordingly. For instance, consider a rod (i.e., one dimension) that permits two bodies (i.e., spheres) to slide along it. The rod is 1 meter in length. Furthermore, we will assume that two bodies can occupy the same location on the rod. Call the position of body 1: x_1, and the position of body 2: x_2. We can then imagine the space defined by $[0, 1] \times [0, 1]$ to be the state space in which we will find points (x_1, x_2) representing the combined position of each of the two spheres. If we

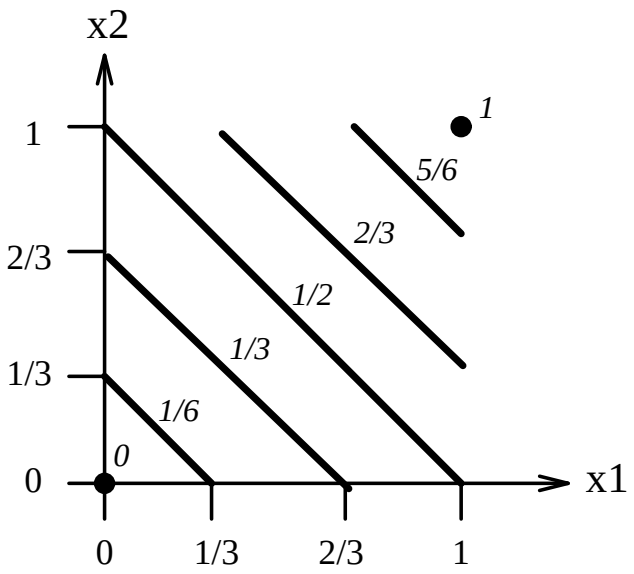


Figure 5: Selected diagonal line partitions of state space.

aggregate the positions of the two spheres by calculating the center of mass, the center mass value can be seen as partitioning the state space with partitions that are diagonal lines. Figure 5 displays the diagonal lines defining constant center of mass for masses that can be positioned at: 0, $1/3$, $2/3$ and 1 along the rod. In the normal situation where the spheres can be placed anywhere (in a continuum) along the rod, the diagonal lines would completely partition the state space.

For this example just discussed, we have accurately represented state space as being a cross product of individual mass locations. However, it is often convenient to let the state space be equal to the state space for a single object, where multiple objects simply overlay the existing state space rather than to add to the dimensionality of the state space. In this case, one can easily partition state space using a clustering or nearest-neighbor method. For instance, if n particles are moving in a two dimensional space then we can partition the space much as we did in fig. 1(a).

2.3 Refining FSAs and Block Models

To discuss abstraction, we first must have an abstraction hierarchy connecting models of different types. Then, we take one arc in the graph and consider the mapping between the two objects on either end of the arc. In this section, we will cover the method of refining a model of one type into another; however, we will consider only the FSA (as an example of a declarative model type) and the block model (as an example of functional modeling). For instance, in our first refinement, we break down part of an FSA into a second level FSA. This is an example of refining as follows: *declarative* $\rightarrow$ *declarative*. States are labeled as s_j, inputs as i_j, outputs as o_j, functions as f_j, and predicates as p_j where j is an index.

When we use the term "FSA" we are actually referring to an *extended* FSA which can be mapped directly to the normal FSA when the predicates p are based on equalities and inequalities. An FSA accepts only inputs on its arcs. Inequality predicates can be mapped

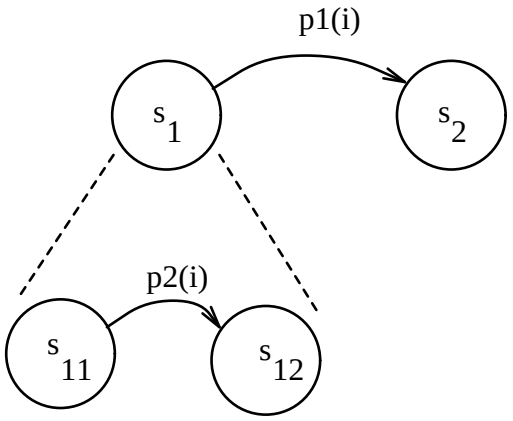


Figure 6: Declarative → declarative.

to inputs in the following manner. Consider an arc from state s_1 to s_2 which has the arc label $x > 3$. In such a case, the variable x is defined in a lower level state space. Moreover, the inequality can be mapped to an "input" by creating a new input variable j and setting $j = 0$ when $x < 3$, $j = 1$ when $x = 3$ and $j = 2$ when $x > 3$. In this way, $x < 3$ serves to label an *internal event* since the input, j, that will cause the state change (to s_2) comes, not externally from the FSA abstraction level, but instead from a lower abstraction level which has a state space including the variable x. We will formalize this concept with $DEVS'$ in section 5.

Consider the FSA in figure 6. This model, along with the one in figure 7, involve *homogeneous* decompositions since one model is defined in terms of models of the same type. Here state s_1 is "broken down" into another FSA at level two which has two states s_{11} and s_{12}. The predicate $p1(i)$ means that there is a predicate involving input variable i such as $i = 0$ ($p2(i)$ is then another predicate such as $i = 1$ if the input is binary valued). Figure 7 defines a two-level functional hierarchy where function f is defined in terms of a composition of three other functions f_1, f_2 and f_3. In functional decomposition, one sometimes increases the dimension of the input and output sets (thereby using our second rule for creating refinements). For instance, suppose that $i_1 \in \mathcal{Z}$. Then, the decimal input can be broken down into a binary input which would involve more inputs to f_1. *Heterogeneous* decomposition of models involves describing the semantics of a model in terms of different model type semantics. For instance, in figure 8, we have an FSA that defines the internal state transitions associated with function f. Indeed, all FSAs can be thought of as having this kind of functional box around them — even though the functional box is not explicit in the FSA graphical representation. The predicate $p(i)$ involves the test of input variable i. Imagine, for instance, that f is the functionality associated with a light switch that "remembers" its current state (ON or OFF). i represents someone flicking the light switch, so $i \in \{TURN_OFF, TURN_ON\}$, s_1 and s_2 mean "light is off" and "light is on," respectively. Then $p(i)$ is defined as $i = TURN_ON$. The arc coming out of state s can be *external* where the predicate involves the i, or it can be *internal* where the predicate involves the use of a state variable (o) connected with the internal block model. Figure 9 displays a state s whose semantics are defined in terms of a functional block model with two functions f_1 and f_2. State s is assumed once $p1(i)$ evaluates to true. At this point the

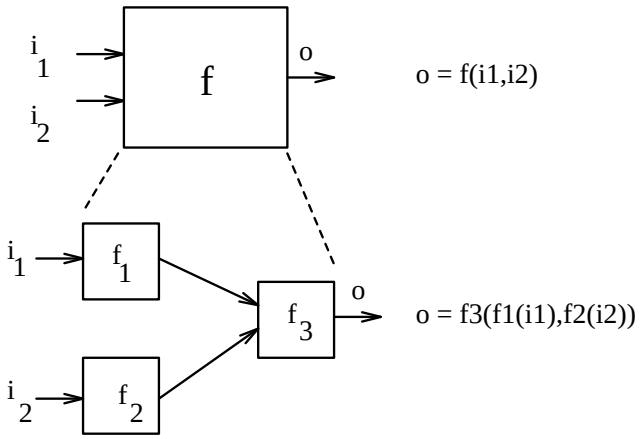


Figure 7: Functional → functional.

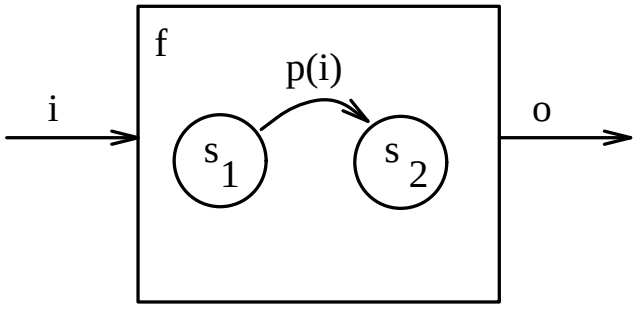


Figure 8: Functional → declarative.

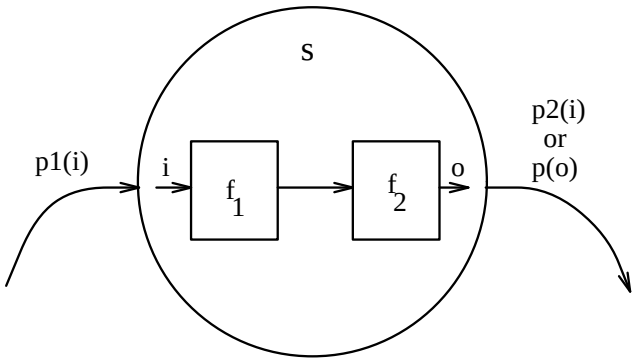


Figure 9: Declarative → functional.

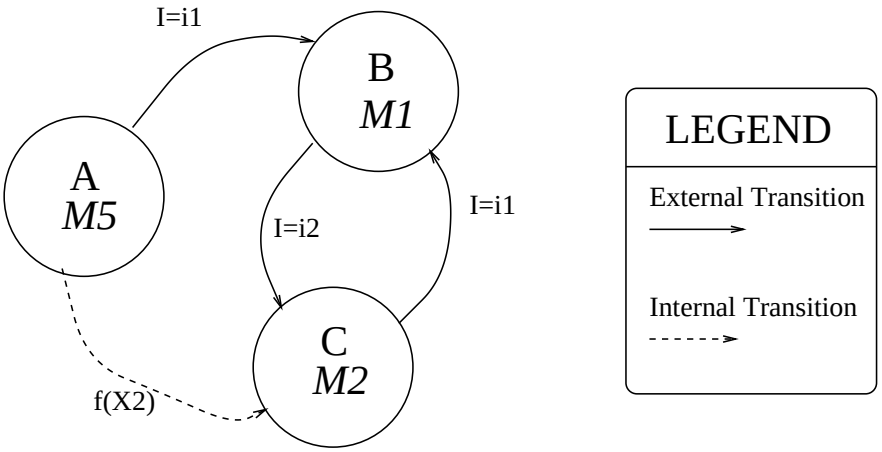


Figure 10: Generic phase graph.

value of i into f_1 becomes the value of i causing $p1(i)$ to become true. Fig. 9 involves a state to state space mapping which is not immediately apparent from the figure. Specifically, most functional block models that represent some aspect of physical reality involve state transitions. This means that f_1 and f_2 contain internal state transitions as shown in fig. 8. The state space for the block model within state s has its own state space — call it X. Given the partition method of abstraction, we will associate s with a partition of the state space X by creating a homomorphic mapping from X onto S where $s \in S$ (i.e., S is the state space of the higher level FSA model whose states contain the block models).

Also note that there is a transition with two possible types of semantics coming out of state s in fig. 9. An external transition would be of the form $p2(i)$ since it is a predicate involving the input i. An internal transitions would be based on a variable that is part of state space X. Most likely, there would be a test (i.e., using a predicate) involving the final output of the block model, $p(o)$, as shown in the figure.

Let's consider a more detailed example involving a refinement as shown in fig. 8. Figure 10 displays a phase graph with 3 phases A, B, and C. The word "phase" simply refers to a state that has lower level semantics attached to it.

A different model is associated with each phase; for instance, in phase B, the underlying model M_1 will generate the state trajectories. External events are denoted in fig. 10

with a solid arc. If in phase 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. 10, 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 on the continuous model level.
- f is an arbitrary predicate with a true or false value.
- M_i is a lower level continuous model containing transition functions defined on state variables X_j.

3 Example 1: Industrial Plant Control

Industrial plants (chemical, oil, nuclear) have fairly complicated systems. These systems involve the sensing of temperature and pressure. When either the temperature or pressure increase beyond certain limits, it is important for the plant operator to either correct the problem or shut down part or all of the system. Consider the following aspects of the control procedure for the plant:

- Assume one sensor for pressure and temperature.
- A normal operating range is specified for pressure ($P_{min} < P < P_{max}$) and temperature ($T_{min} < T < T_{max}$). When a sensor detects a value out of this range, an alarm sounds and a signal is raised.
- If, after a specified period of time K, there is a successful recovery then the plant automatically returns to normal operation. Otherwise, the plant is shut down.

Figure 11 displays an FSA which controls the plant. The following symbols are used in fig. 11: T (temperature), P (pressure), T_r (reference[4] temperature, P_r (reference pressure),

[4]The term "reference" refers to the control value set by the external influence (i.e., the operator).

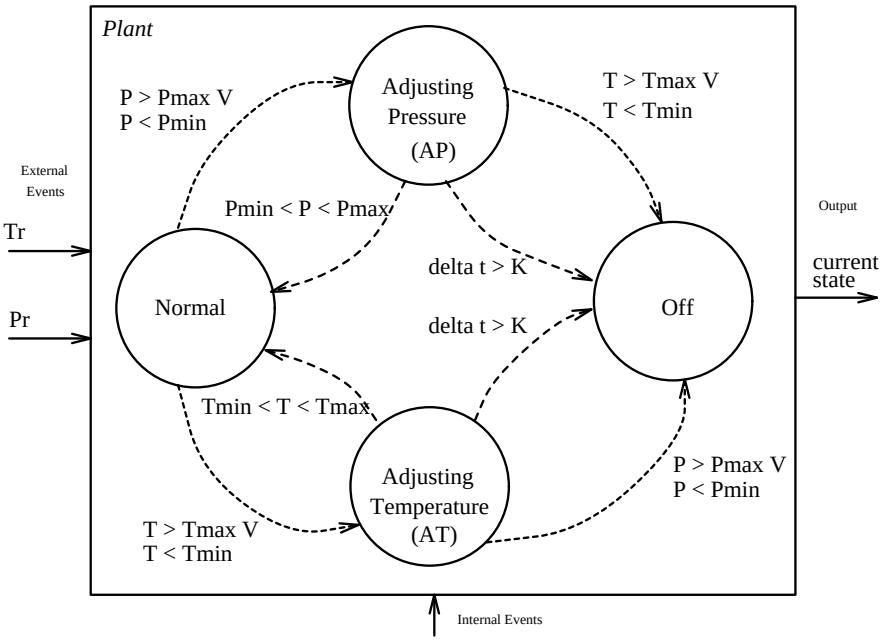


Figure 11: Industrial plant control multimodel.

ΔT (amount of time passed in the current state), K (amount of time before plant shutdown) and the minimum and maximum values defining the range for temperature (T_{min},T_{max}) and pressure (P_{min},P_{max}).

Each of the four basic states has lower level semantics (i.e., another abstraction level). For instance, consider the state of "adjusting temperature." It can be refined into a block model defining a proportional method of control shown in figure 12.

The phase space after partitioning is shown in figure 13 for the space $T \times P$.

4 Example 2: A Parking Lot Attendant

Consider the actions of a parking lot attendant. A parking attendant managing a 100 car parking lot is responsible to see that people pay the meters. Every hour (on the hour) the attendant visually checks every meter to see if it is still running. A simulation of the attendant's actions over time could be modeling by a difference equation or a finite state model with a loop; however, we now consider that an attendant may, at any time, be stopped to answer two types of questions: 1) "What is the time?" and 2)"Where is store XYZ?". We say that, aside from answering these questions, an attendant will be doing one of two things (i.e., the attendant is in one of two states): *checking* cars, or *sitting*. Here are the times for the states as specified:

1. Answering time question: 5 seconds.
2. Answering locative question: 30 seconds.
3. Checking cars: 20 minutes.

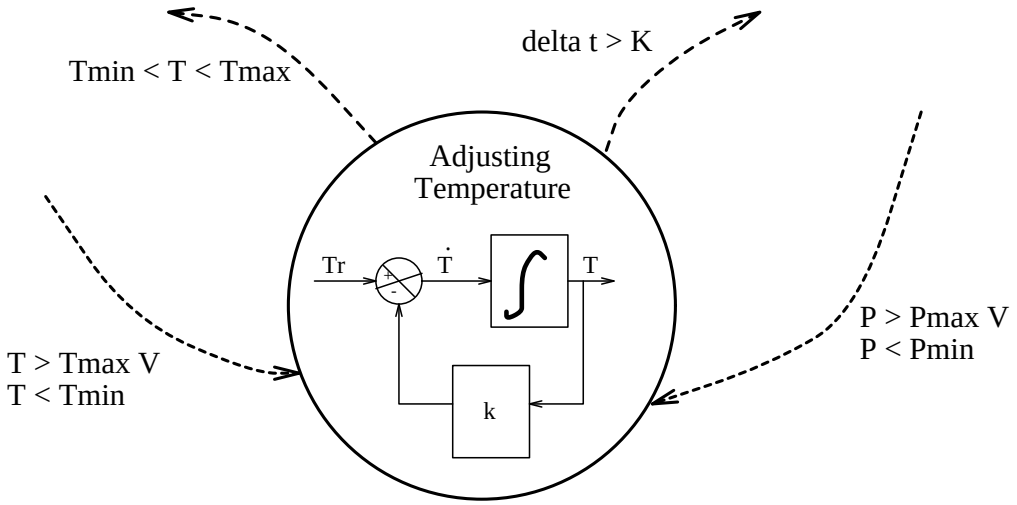


Figure 12: Decomposition of "adjusting temperature" state.

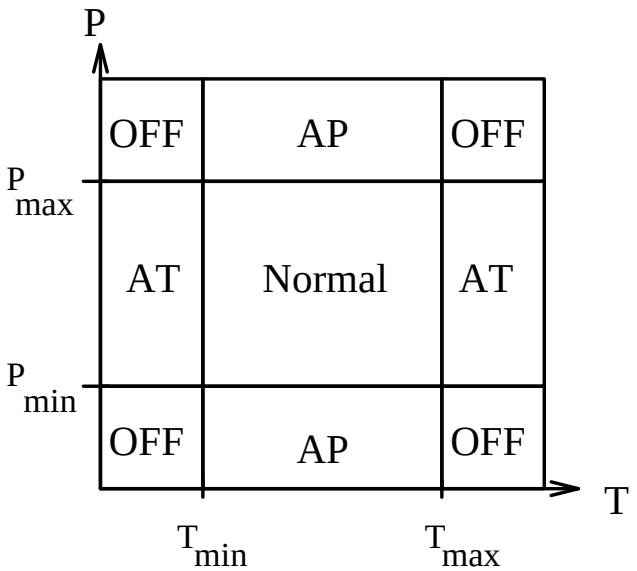


Figure 13: Partition of phase space for the plant.

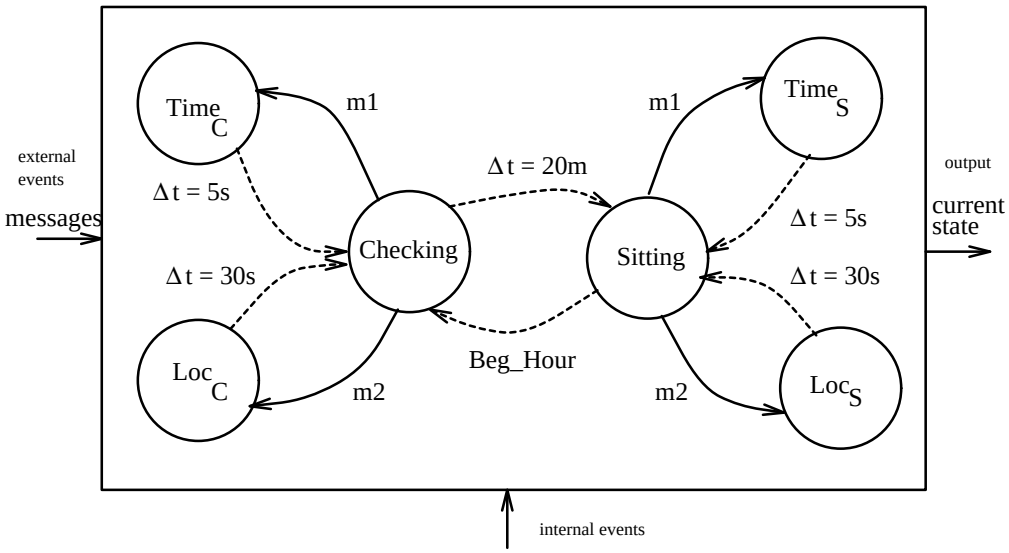


Figure 14: Parking attendant dynamics.

4. Sitting: remainder of time not spent doing 1 to 3 above.

Figure 14 displays our model. We now think of the states "checking" and "sitting" as being different locations within a lower level state space X^2. We will design two rest areas for sitting and an area for the parking lot (see figure 15. In this case, partitions refer approximately to (x, y) vertices of the rectangles defining the partitions. Therefore, the attendant would make small linear movements while staying in the parking lot partition ($20 < x < 380$ and $100 < y < 370$). The linear motion is represented a lower level dynamical model whose state variables are x and y.

5 DEVS' System Formalism

$DEVS$ (Discrete Event System Specification) [19] is a formalism that combines elements of declarative with functional simulation modeling. The formalism permits one to construct functions which respond to internal state transitions and external inputs. $DEVS$ provides a formal way to create a particular form of two level multimodel: a coupled network of functional block models where each block can contain a set of internal transitions. For our purposes, we limit the internal transitions to those that can be performed with an FSA. We modify the $DEVS$ definition slightly to explicitly incorporate the concepts of internal event *conditions* and to de-emphasize the explicit notation for elapsed time[5]. We will term this new kind of model structure $DEVS'$ which serves as a formalism that is similar to the $DEVS$ formalism. $DEVS'$ bears a significant resemblance to Praehofer's $DEVS\&DESS$ formalism [12, 13] which was constructed to extend DEVS for combined modeling. The purpose behind the construction of $DEVS'$, though, is to show how *one type*

[5] Elapsed time is assumed to be calculated correctly (or "unambiguous") based on the time when 1) external inputs interrupt the system in some state, or when 2) internal state events occur to cause a state change.

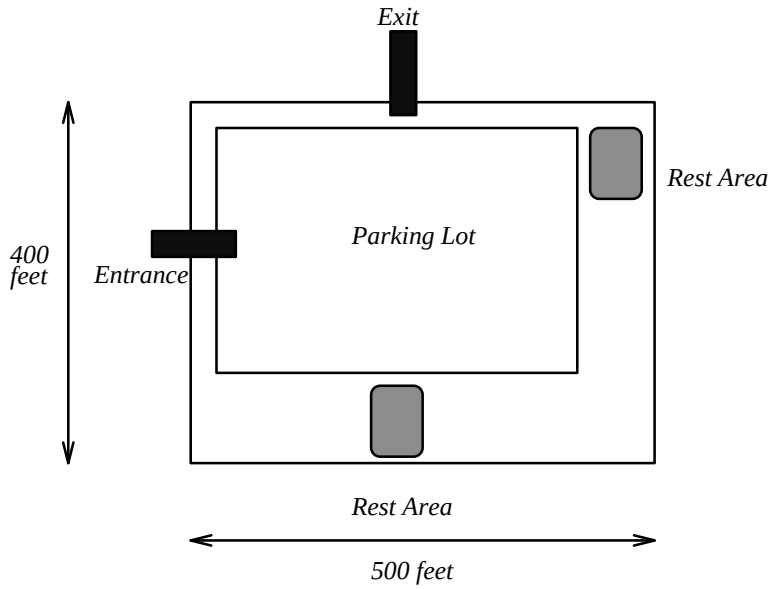


Figure 15: Geometric phase partition for the parking area.

of multimodel— namely, one where the model is a functional block model with internal state transitions of the extended FSA variety— compares with the $DEVS$ formalism.

$$DEVS' =< X_{int}, X_{ext}, Y, S, \delta_{int}, \delta_{ext}, \lambda >$$

where:

1. X_{int} is a set of internal inputs from the next lower abstraction level.
2. X_{ext} is a set of external inputs from the current abstraction level.
3. Y is a set of outputs.
4. S is a set of states.
5. $\delta_{int} : S \times X_{int} \rightarrow S$. δ_{int} is the "internal" transition function which allows for *state events* or *time events*. Time events are events that occur at a specific time regardless of what is input to the system. Both time and state events occur "internal" to the system. On a graph of the model, internal events are denoted by predicates which are evaluated in a lower level (i.e., of abstraction) process. Note that in $DEVS$, the internal state transition function is defined differently as $\delta_{int} : S \rightarrow S$ where a time advance function ta provides the amount of time spent in a state assuming no external events occur.
6. $\delta_{ext} : S \times X_{ext} \rightarrow S$. δ_{ext} is the "external" transition function which allows for input events. Input events are events which occur as a result of entities arriving as inputs from outside of the system (i.e. arrivals into a queue or parts on an assembly line) at the current abstraction level.
7. $\lambda : S \rightarrow Y$. λ is the output function.

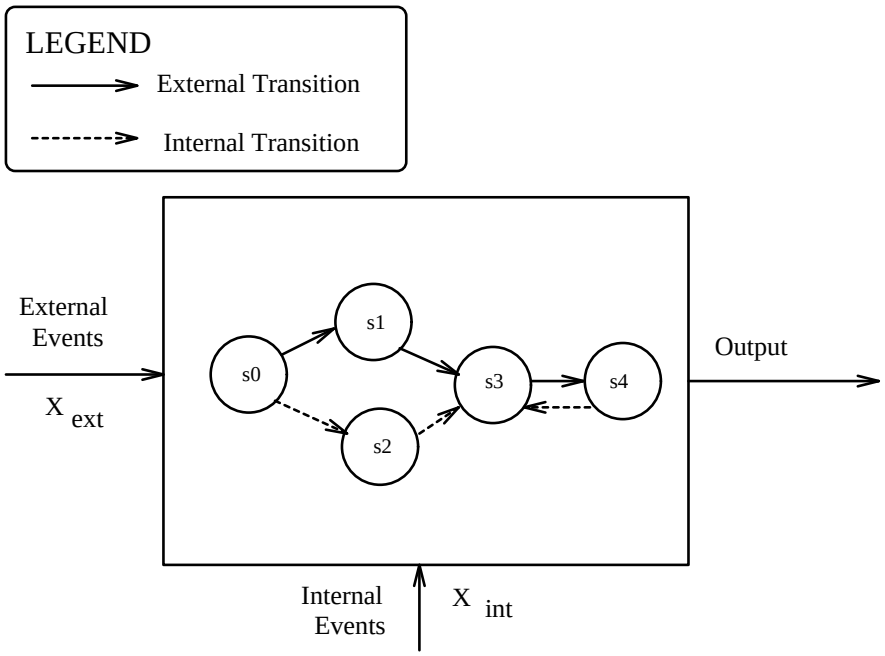


Figure 16: General structure of a $DEVS'$ model.

The $DEVS'$ model captures declarative concepts in the form of the *internal transition function* and functional concepts in the form of the *external transition function*. Graphically, one may depict a $DEVS'$ model illustrated in figure 16.

$DEVS'$ views a model as a coupled network of individual blocks. Data or tokens flow through the network much as they do in a functional block network. The primary addition in $DEVS'$ is the concept of having internal as well as external transitions. Internal transitions are denoted with a dashed arc, and external transitions use a solid arc. We begin with a simple example of a $DEVS'$ model and then formalize the earlier discussion of the parking lot attendant dynamics.

5.1 DEVS' Example 1: A Single Server Queue

Let's consider the example of a single server queue and six customers where the inter-arrival times are 20,30,25,65,80 and the service times are 80,150,90,60,80. The state of our system will be dependent upon how many customers are in the system (this includes both the queue and the server). A $DEVS'$ model for this scenarios is as follows:

$$DEVS' =< X_{int}, X_{ext}, Y, S, \delta_{int}, \delta_{ext}, \lambda >$$

where:

1. $X_{int} \in \{d\}$ where d is a departure from the system. Departures are determined by an "internal event" identified by a length of time (time event) or a state value crossing a threshold (state event). The semantics associated with this event are not listed here; however, they are often in the form of a difference equation defining the new departure time as a function of last arrival, last departure and service times.

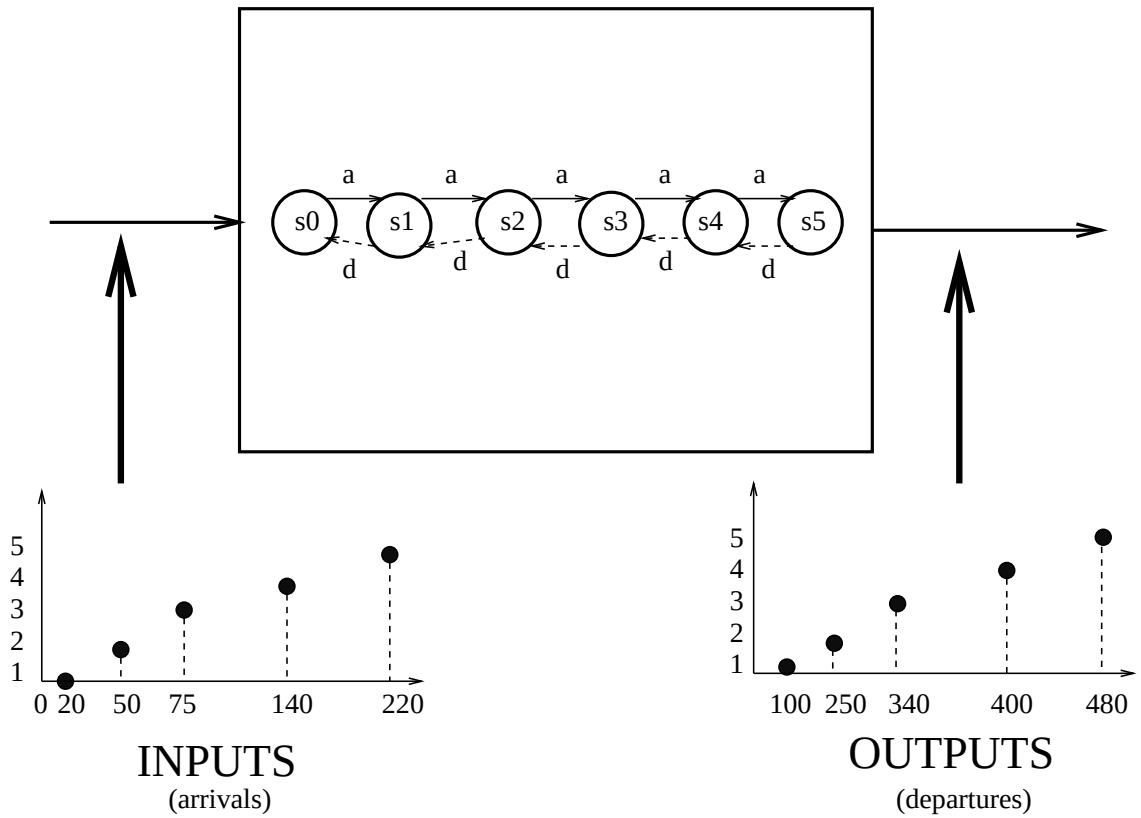


Figure 17: Graphical model of $DEVS'$ for the single server queue example.

2. $X_{ext} \in \{a\}$ where a is an arrival into the system. Arrivals come at the specified time intervals.

3. $S \in \{s_0, \ldots\}$ where s_i is the state where i customers are in the system.

4. $\delta_{int} : S \times X_{int} \rightarrow S$ is defined as: for i = 1 to 6: $\delta_{int}(s_i, d) = s_{i-1}$.

5. $\delta_{ext} : S \times X_{ext} \rightarrow S$. δ_{ext} is defined as follows: $\delta_{ext}(s_0, a) = s_1$, $\delta_{ext}(s_1, a) = s_2$, $\delta_{ext}(s_2, a) = s_3$, $\delta_{ext}(s_3, a) = s_4$, $\delta_{ext}(s_4, a) = s_5$ and $\delta_{ext}(s_5, a) = s_6$.

6. $\lambda : S \rightarrow Y$. λ is the output function.

The external transition function is responsible for defining when an input causes the system to change state. The only input to this system is a customer arriving at a specific time: a discrete time trajectory with arrivals. An internal transition function is responsible for defining when an internal event —as caused by checking an internal state value— causes the system to change state. Departures are due to internal state changes; the simulator keeps track of length of time spent in the queue and undergoing service and then causes an internal event when a departure is to occur. Departures, like arrivals, are discrete time trajectories leaving the system.

Figure 17 displays a graphical model of the $DEVS'$ specification for the single server queue.

5.2 DEVS′ Model of the Parking Lot Attendant

A $DEVS'$ model for the parking attendant dynamics is defined as:

$$DEVS' =< X_{int}, X_{ext}, Y, S, \delta_{int}, \delta_{ext}, \lambda >$$

where:

1. $X_{int} \in \{Time_C\Delta t, Loc_C\Delta t, Checking\Delta t, Sitting\Delta t, Time_S\Delta t, Loc_S\Delta t\}$. These "delta" times are specified for each type of internal event.
2. $X_{ext} \in \{m_1, m_2\}$. m_1 and m_2 are message types 1 (time related query) and 2 (location related query).
3. $Y = S$.
4. $S \in \{Time_C, Loc_C, Time_S, Loc_S, Checking, Sitting\}$
5. $\delta_{int} : S \times X_{int} \rightarrow S$. A condition becomes true when an internal state variable (such as elapsed time) becomes equal to a certain threshold. $\delta_{int}(Checking, Checking\Delta t) = Sitting$. $\delta_{int}(Sitting, Sitting\Delta t) = Checking$. $\delta_{int}(Time_C, Time_C\Delta t) = Checking$. $\delta_{int}(Loc_C, Loc_C\Delta t) = Checking$. $\delta_{int}(Time_S, Time_S\Delta t) = Sitting$. $\delta_{int}(Loc_S, Loc_S\Delta t) = Sitting$.
6. $\delta_{ext} : S \times X_{ext} \rightarrow S$. δ_{ext} is defined by discrete state input trajectories that specify when messages of type 1 and 2 are entered into the system.

6 Conclusions

Multimodeling provides one way of dealing with complex system modeling — namely, treat the system as a network of abstractions where each abstraction is tied to another using state space partitioning and homomorphic mapping. We have promoted the use of geometric partitioning of phase space as an approach to creating abstract models from more detailed models whose state variables are defined on the lower level —often continuous— state space. Most systems (especially nonlinear ones) provide natural partitions of state space in the form of basins of attraction, whereas others map our cognitive model of a system into partitions of a lower level state space. We have demonstrated that by creating multimodels, we can both control and organize lower level, more detailed, systems whose state spaces are effectively partitioned. It is still unclear how to automatically derive such partitions; however, we are investigating the use of clustering methods, and other concept formation schemes in artificial intelligence.

References

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

[2] COOK, P. A. *Nonlinear Dynamical Systems*. Prentice Hall, 1986.

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

[4] FISHWICK, P. A. The Role of Process Abstraction in Simulation. *IEEE Transactions on Systems, Man and Cybernetics 18*, 1 (January/February 1988), 18 – 39.

[5] FISHWICK, P. A. Abstraction Level Traversal in Hierarchical Modeling. In *Modelling and Simulation Methodology: Knowledge Systems Paradigms*, B. P. Zeigler, M. Elzas, and T. Oren, Eds. Elsevier North Holland, 1989, pp. 393 – 429.

[6] FISHWICK, P. A. Heterogeneous Decomposition and Coupling for Combined Modeling. In *1991 Winter Simulation Conference* (Phoenix, AZ, December 1991), pp. 1199 – 1208.

[7] FISHWICK, P. A., AND ZEIGLER, B. P. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation 2*, 1 (1992).

[8] NICOLIS, G., AND PRIGOGINE, I. *Exploring Complexity: An Introduction*. W. H. Freeman, San Francisco, 1989.

[9] OREN, T. I. Dynamic Templates and Semantic Rules for Simulation Advisors and Certifiers. In *Knowledge Based Simulation: Methodology and Application*, P. Fishwick and R. Modjeski, Eds. Springer Verlag, 1991, pp. 53 – 76.

[10] PADULO, L., AND ARBIB, M. A. *Systems Theory: A Unified State Space Approach to Continuous and Discrete Systems*. W. B. Saunders, Philadelphia, PA, 1974.

[11] PALATNIK, L. S., AND LANDAU, A. I. *Phase Equilibria in Multicomponent Systems*. Holt, Rinehart and Winston, Inc., 1964.

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

[13] PRAEHOFER, H. Systems Theoretic Formalisms for Combined Discrete Continuous System Simulation. *International Journal of General Systems 19*, 3 (1991), 219–240.

[14] PRIGOGINE, I. *From Being to Becoming: Time and Complexity in the Physical Sciences*. W. H. Freeman, San Francisco, 1980.

[15] PRITSKER, A. A. B. *The GASP IV Simulation Language*. John Wiley and Sons, 1974.

[16] PRITSKER, A. A. B. *Introduction to Simulation and SLAM II*. Halsted Press, 1986.

[17] ZEIGLER, B. P. *Theory of Modelling and Simulation*. John Wiley and Sons, 1976.

[18] ZEIGLER, B. P. DEVS Representation of Dynamical Systems: Event-Based Intelligent Control. *Proceedings of the IEEE 77*, 1 (January 1989), 72 – 80.

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