Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

GRAPHIC MODELING USING HETEROGENEOUS HIERARCHICAL MODELS

Victor T. Miller
Paul A. Fishwick

Department of Computer and Information Sciences
University of Florida,
Gainesville, Florida, 32611, U.S.A.

ABSTRACT

The importance of efficiently representing each abstract level of a complex model has become significant for many reasons. Because of the size of complex models, an organizational methodology is needed as an aid during both development and investigation of a system. In addition, some modeling formalisms are design to describe certain classes of systems while others are general enough to represent a broad class of systems. Therefore, the ability to use multiple formalisms is advantageous when the model is complex.

In this paper, we define *Heterogeneous Hierarchical* modeling (HH modeling) as a modeling methodology which allows an investigator to use multiple formalisms and to organize these formalisms hierarchically. We then present *Hybrid Model Theory* which supports HH modeling by theoretically unifying graphic-based formalisms such as Petri nets, block models and state machines. An example is given which demonstrates top-down refinement of a partial model using a variety of established, well-known formalisms.

1 INTRODUCTION

Heterogeneous hierarchical modeling is defined by Miller (1993) to describe any modeling methodology which supports several conceptually distinct representations and hierarchical development of system simulations. Additionally, a unified theory is needed in order to provide a firm mathematically-based foundation upon which one can build a method to improve the conceptual and developmental efforts of an investigator. The unified theory is not to be used by the "end user" of a computer simulation modeling environment, but rather the theory is meant only to provide a formal medium which allows a computer environment to automatically perform certain functions on behalf of the user. In particular, the formal theory must stipulate how heterogeneous modeling formalisms (e.g., Petri nets, block models) can be used hierarchically to form a single system model.

An investigator who needs to create a large complex system model requires the ability to choose a formal representation which is most appropriate to model the subsystem currently under consideration and requires an organizational structure to formally compose the subsystems into a single system. To this end, hierarchical modeling provides a representation that organizes the model for both the investigator and the computer environment in which it is developed. It also allows for incremental refinement of a model or a portion of a model in top-down development. It also furnishes a structure for bottom-up composition from previously established models which may be in a database.

In this context, there are two categories of model hierarchies which should be distinguished: *type-of* and *part-of* hierarchies. Type-of hierarchies are related to object-oriented models and are usually the focus of the software engineering and artificial intelligence communities. However, object-oriented models also appear frequently in the simulation literature (Nelson 1991). Type-of hierarchies emphasize the categorization of entities based on the generalization of static properties. Part-of hierarchies can describe either static or dynamic properties and emphasize the categorization of physical or conceptual composition. Although both types of hierarchies are essential, we limit our discussion to part-of hierarchies.

When using part-of hierarchies, there are two methods in which the hierarchy is constructed. These two methods roughly relate to bottom-up and top-down development. An *intermodel* hierarchy defines the coordination of input and output *between* formal models and, therefore, favors bottom-up development of abstract models by grouping preexisting formal models. The DEVS system (Zeigler 1984) is a good example of intermodel coordination (however, DEV's system entity structure is a type-of hierarchy). An *intramodel* hierarchy defines the coordination of input and output from *within* a formal model to another different formal model and, therefore, favors top-down refinement of an abstract model into submodels. An HH modeling theory must, in general, support both types of coordination in order to allow the maximum flexibility to an investigator. We present an intramodel hierarchy in section 3 of this paper.

In an accordant relationship with hierarchical modeling is heterogeneous modeling. Heterogeneous modeling refers to any type of representation which permits the integration of a diverse set of modeling formalisms (such as knowledge-based, discrete, continuous, fuzzy, and object-oriented models) into a single formalism; hence heterogeneous modeling expands the definition of discrete-continuous modeling.

In heterogeneous modeling there are two categories of model classes: static and dynamic. Static model formalisms, (e.g., semantic nets) are used to describe entity types and are used in type-of hierarchies or graphs. Although there is no theoretical impediment to prevent static model formalisms from including dynamic information, dynamic model formalisms (e.g., Petri nets and queuing nets) are generally used to describe the time depend relationships among entities in a system.

In order to support HH modeling, *hybrid model theory* was recently purposed as a possible formal representation (Miller 1993). Hybrid model theory originated from Fishwick and Zeigler's multimodel methodology (1992) and is an alternative approach to other such modeling formalisms as hierarchical, modular models (Zeigler 1990) and combined discrete-continuous system simulation (Praehofer 1991).

Hybrid model theory is a specialized construct that is defined in terms of general system theory and permits a single model to be hierarchically constructed either by inter or intramodel coordination from five well-known formalisms: Petri nets, Markov systems, queuing networks, state machines and block models. The hierarchy is based upon the composition of a system (part-of hierarchy) rather than the classification of entities as purposed in object-oriented simulation (e.g., Simula, C++ libraries). Although some primitive type-of information can be derived from a part-of hierarchy, the structure attempts to encapsulate only the dynamic behavior of a system within the hierarchy.

In hybrid model theory, state formalisms model the transition of a system from one discrete event to another (state machines, Markov systems), selective formalisms model events based on resource allocation (queuing networks, Petri nets) and functional formalisms model continuous signal-based systems (block models). These combined formalisms, together with a hierarchically organized development method, increase the ability of an investigator to construct system models in a productive manner.

2 HYBRID MODEL THEORY

Although hybrid model theory (HMT) encapsulates several representations, only those properties of a hybrid model which directly relate to simulation are presented. In hybrid model theory a state machine, Petri net, etc., are not atomic models but structured models. Structured models are made up of at least two hierarchical levels. The first level is called a *controller* model. For a variety of formalisms only four controller models are necessary. The second level is made up of atomic models called *component* models. This split-level approach to HMT is demonstrated in Figure 1.

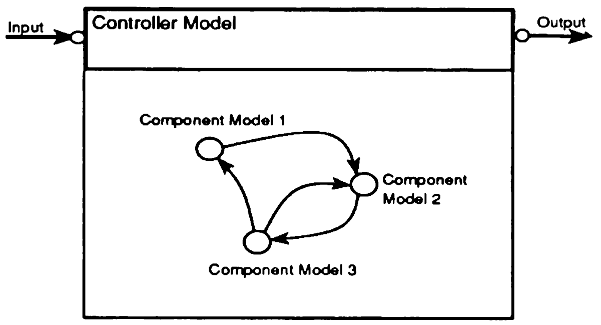


Figure 1: HMT Representation of a Model

As shown in the figure, the data (or control) "Input" and "Output" are directed into the controller model. The component models (nodes in the graph) may or may not have data input and output. Depending on the type of controller, edges in the graph will either indicate control flow or data flow. This dual functionality has been captured in the controller model's interpretation of its components (via the transition relation to be described next). Only under direction of the controller model is data passed down to and up from the component models or submodels. In hybrid model theory, refining a component model into another modeling formalism is called submodeling.

Formally, a hybrid model M is a named set such that M = $<H, A, X, \Psi, \Theta, \tau, \beta, \delta, \mu, \lambda>$ and

H: Component	$<self , \eta_1, ...>$
A: Edge	$<\alpha_1, \alpha_2, ...>$
X: Input	$<\chi_1, \chi_2, ...>$
Ψ: Output	$<\psi_1, \psi_2, ...>$
Θ: State	$<\theta_1 ,\theta_1, ...>$
τ: Time Domain	$<(T,+,\leq)$, z, d, m[], mag[]>
β: Initialize	$(\tau, \chi, \theta) \rightarrow \theta$
δ: Transition	$(\tau, \chi, \theta) \rightarrow \theta$
λ: Output	$(\tau, \chi, \theta) \rightarrow \psi$.

The symbols < and > indicate the use of named sets, and elements in these sets can be accessed as described by Zeigler (1984). The special symbol *self* is always a member of the component set (H) of a model. This symbol is used to indicate a reference to a submodel (if one exists). The self symbol is the only member of the component set for component models. In controller models, the component set contains the models which are supervised by the controller model. The edge set (A) of a component model is empty. In controller models, the connectivity between component models is identified with the edge set. An edge $\alpha \in A$ is a named set of the form <*to, from, type*>, where the symbols *to* and *from* are models in the component set (H), and type is either undefined (†) or a structured data type based on the reals ($\Re$)or the integers ($\Im$). Together, the components and edges describe the graph of the model and either what type of data is passed between the components or how control is transferred among the components.

The input (X) and output (Ψ) also have the form <*to, from, type*>. These sets signify the data or control information used by different types of models. There is one difference between inputs and outputs; Inputs are signals (functions over time) and outputs are values. The state (Θ) named set is used for a variety of purposes, depending on the type of controller or component model. It is very similar to local memory in computational definitions.

The time domain τ of a model is used in knowledge-based simulation and is not essential to the concepts in this paper. It is only noted here that the time domain of a model contains the time quantum used in numerical-based simulations.

The last three elements of a model are relations. Typically, these are used to compute the new state and output trajectories over a time interval. Because hybrid model theory is centered around simulation concepts, these relations have been conceptually altered. It is assumed that all three relations use two times: the current time (a global variable) and an end time (given at relation invocation). These times are used to calculate the state or output at the end time. The current input and state are also assumed to be part of the input to these relations. Output trajectories are created by symbolic methods which take a model hierarchy as input or created through numerical analysis techniques. Additionally, it should be emphasized that these relations are declared, not precompiled. When numerical analysis (simulation) is needed, the declarative model is compiled and optimized.

The initialization relation (β) is necessary since models can be dynamic. At any time during analysis, a model can become active. This not only allows for the modeling of systems which may lay dormant, but more importantly, it models systems which have multiple descriptions over time. The transition (δ) relation is intended to be used when a model is active. Although, as can be seen from the description, it could be used to get the initial state of a model. The initialization and transition relations were derived so that the concept of state, transition and initialization could be separated. For the same reason (and tradition), the output relation (λ) is also kept separate from the other relations.

One of the optimizations for numerical analysis is the integration of the three relations (β, δ, λ) so that only one invocation of the model's relations produces the total behavior. This integration is possible in hybrid model theory because there are only four controller models and each type of controller has the same form of transition and output relations. For instance, the difference between a Markov system and a state machine is the component models. The controller model is the same.

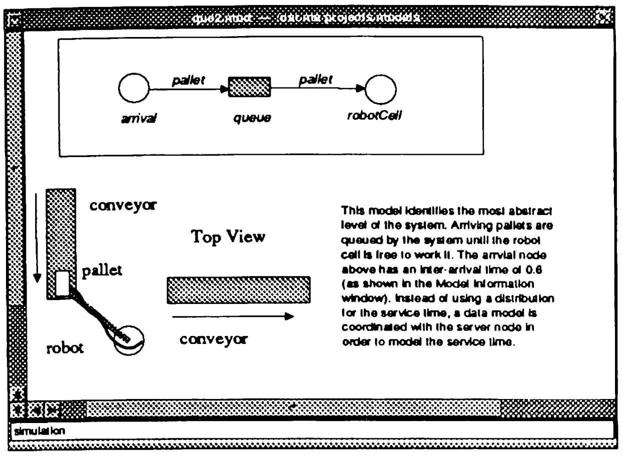
3 EXAMPLE

A modeling environment has been developed to demonstrate and verify hybrid model theory. Figure 2 shows the graphical interface to the environment which allows an investigator to draw the simulation using a selected formalism. It also allows the investigator to sketch the objects or concepts being simulated and to add text comments to the drawing (the simulation package was developed from an object-oriented diagram drawing package).

In this example, a simulation of a robot cell is modeled with a queuing network. The robot cell simply transfers a pallet from one conveyor to another. The simulation is represented by the small graph at the top of the window. Everything else in the window is considered as text or graphical comments. Another window (called an inspector) allows the investigator to select details about the selected object in the window. For text and graphic comments, the investigator selects features such as fill color, line width, etc. For simulation objects, the investigator selects properties such as distributions, signal type (real, integer), queue capacity, trace options, etc.

The HMT representation of this simple queuing network is

H: <self , arrival, queue, robotCell>
A: <<arrival, queue, †>, <queue, robotCell, †>>
X: <>
Ψ: <>
Θ: <queueState>
τ: <($\Re$,+,$\leq$), z, d, m[], mag[]>
β: (τ, χ, θ) -> <queueState = idle>
λ: (τ, χ, θ) -> <>.



This model identifies the most abstract level of the system. Arriving pallets are queued by the system until the robot cell is free to work it. The arrival node above has an inter-arrival time of 0.6 (as shown in the Model Information window). Instead of using a distribution for the service time, a data model is coordinated with the server node in order to model the service time.

Figure 2: GUI to Modeling Environment

The transition relation is far more complex, but is described in detail in Miller (1993). Essentially, it computes the state at each new event (arrival or departure). This continues recursively until the end time is reached. For numerical analysis (simulation), the recursion is compiled into an event loop.

Once the simulation is drawn and the details set in the inspector window, the investigator can select the "compile and run" option in the pull down menu and the program will compile the model into code, run it, and present the investigator with a list of the traces that he/she selected in the model. The investigator then has the option of selecting one or more traces and viewing them graphically or displaying statistics about them. Figure 3 shows a sample plot of the robot cell's busy state when the arrival node has an exponential arrival time of 0.6 and the robot has an exponential service time of 0.5.

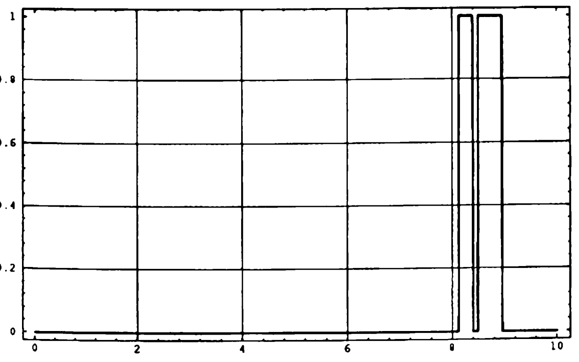


Figure 3: - Plot of Robot Cell Busy State vs Time
1 = busy, 0 = idle

This simple queuing network model can be used as an abstract model for a more complex model. Figures 4 and 5 show models of a robot master and motor which are stored in a database. The robot master model controls a single degree of freedom robot arm by supplying an output voltage given the robot arm's current angle (Figure 4). As the diagram indicates, a state machine is used to model this system. The robot master also supplies a signal indicating when the robot arm is busy (moving a pallet). Thus, each state must supply 2 output signals (voltage and busy) and uses 1 input signal (angle).

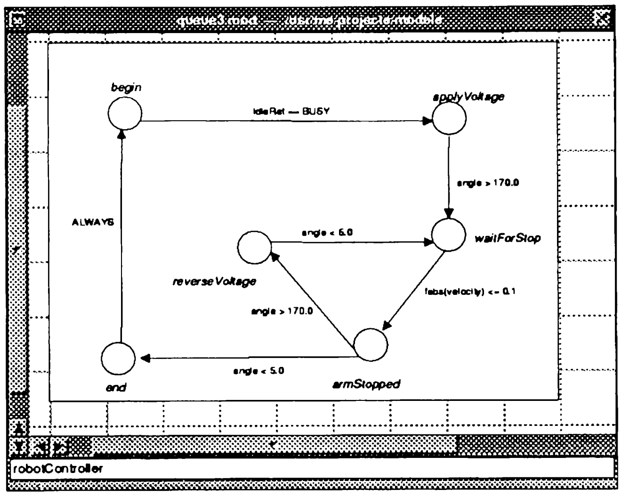


Figure 4: Model of Robot Master States

The motor has been modeled using a block model (Figure 5). This model uses a voltage to calculate the current angle of the motor's shaft. These two models will be used together in an *intermodel* hierarchy to model the time required for the robot cell server in Figure 2 to transfer a pallet from one conveyor to another. These models are coordinated as in Figure 6.

In Figure 6 there are a total of three models which have been coordinated. This type of bottom-up development is similar to DEVS (Zeigler 1984) and combined-discrete simulation (Praehofer 1991). However, hybrid model theory integrates the concept of time somewhat differently that either Zeigler's or Praehofer's work. The robot state component model in Figure 6 is another state machine which calculates the internal state of the robot and is not shown in this paper.

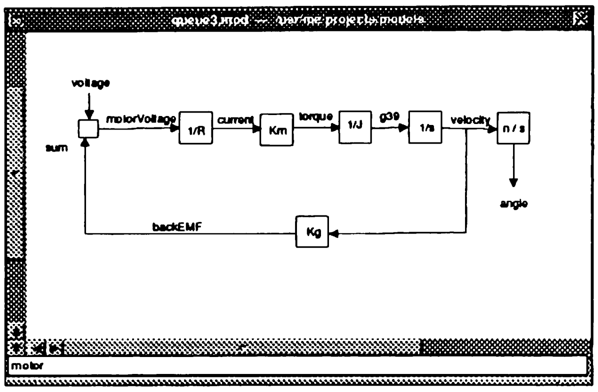


Figure 5: Model of Motor

The model of Figure 6 can now be used to implement the robot cell server in Figure 2 by using an *intramodel* hierarchy. In HMT any model which supplies a "default" idleRet (or busy) signal can be used to refine a transition in a Petri net or a server in a queuing network. The operation is fairly simple. When the queuing model determines that the robot cell server is active, the model of Figure 6 is initialized (hence the need for an initialize relation). The queuing network model monitors the idleRet (or busy) signal. When the signal goes low, the queuing network interprets this as the server being idle. Thus, the server time has been replaced by a more complex model which better represents the actual system. In hybrid model theory, there are a few of these "default" signals which allow coordination between different models. They are very easy to learn, and with them any type formalism can be used to refine a component of another formalism.

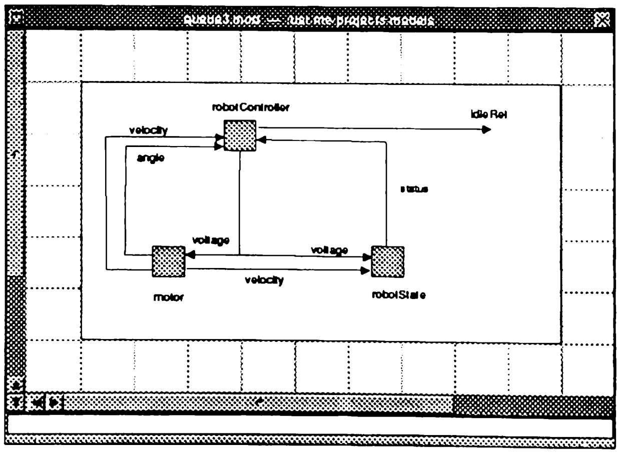


Figure 6: Intermodel Coordination

A numerical simulation of this model can also be compiled and run by the investigator. Figure 7 shows the plot of the three traced signals: voltage, angle and velocity. This simulation run began with one pallet in the queue. As can be seen from the trace, at time zero the queue network initialized the robot cell model and waited until the robot controller return an idle signal. During this time, the voltage applied to the motor was 120 volts from time 0.0 to 3.1 and -120.0 from time 3.1 to 6.2 and was supplied by the state machine. The robot arm's angle moved from 0 degrees to 180 and then back down to 0 again and was supplied by the block model. Also from the trace, it can be seen that more pallets arrived starting at time 33.0. In between time 6.2 and 33.0 the server of the queuing network, the robot cell server, was idle and therefore the robot cell model of Figure 6 was deactivated. In this interval the signals are undefined. At time 33.0 when another pallet arrived, the robot cell model is again initialized and the queuing network waits for the idle signal.

Fuzzy and qualitative simulations (Fishwick 1991) can also be run using the computer environment. In hybrid model theory signals and time domains are structured objects and are stored in a database. When the investigator declares a signal type, he/she selects the type from the database. These structured types contain the necessary information for all three types of simulation. In Figure 8, an example of the signal type database is shown. This information allows the computer environment to check several types of semantic errors (such as out-of-range errors during simulation and dimension analysis). It also allows the use of linguistic values for fuzzy simulation (Fishwick 1991), and it permits the computer environment to set up simple qualitative spaces for the investigator in qualitative simulation (Bobrow 1986).

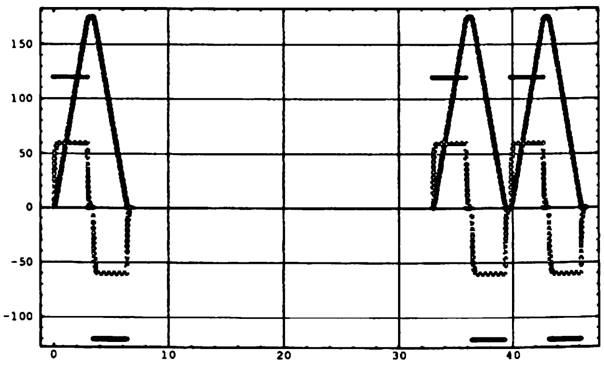
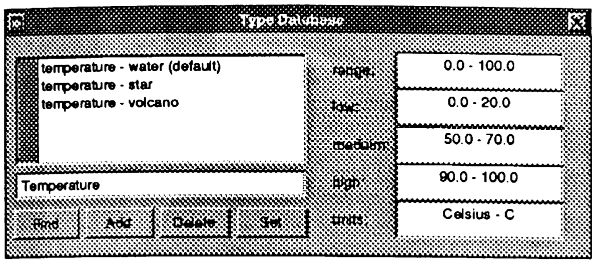


Figure 7: Traced Signals of complex model vs Time. Voltage & Velocity - black, Angle - gray



temperature - water (default)
temperature - star
temperature - volcano

Temperature

range:	0.0 - 100.0
low:	0.0 - 20.0
medium:	50.0 - 70.0
high:	90.0 - 100.0
units:	Celsius - C

Figure 8: Type Database

4 SUMMARY

The example given in the last section demonstrates how an investigator can develop a model using either a top-down or bottom-up methodology. At any point, the investigator can select a formalism which suits the pragmatic issues at hand and coordinate this model within the system. Hybrid model theory dictates how these coordinations must occur in order to maintain consistent input/output, time and model control; nevertheless, for all practical purposes, hybrid model theory is hidden from the investigator. The program ensures correctness by automatically setting up the appropriate signals and coordination, and the investigator is free to concentrate on modeling the system.

There are a few important limitations to the current version of hybrid model theory. Many simulation packages allow a user to model the entities in a queuing network and the tokens in a Petri network. It is reasonable to assume that these entities represent important components of a system. Hybrid model theory, at the moment, does not allow these entities to modeled. However, current research indicates that there is no major obstacles to including this type of modeling.

There are times, especially in larger systems, when a duplicate or slightly modified model is needed in order to implement a new component of the system. Sharing a model between two parts of a system can only occur when the two parts are guaranteed never to be active at the same time; otherwise, a duplicate mode is needed. When a model is needed which is slightly different than an existing model, the investigator must duplicate and then modify the model. An integration of of type-of hierarchies into hybrid model theory is needed for this to occur efficiently.

REFERENCES

Bobrow D.G (ed), Qualitative Reasoning about Physical Systems, MIT Press, Cambridge, Massachusetts, 1986.

Fishwick, P.A., "Fuzzy Simulation: Specifying and Identifying Qualitative Models," International Journal of General Systems, 19 (3), pp. 295 - 316, 1991.

Fishwick P. A., and B.P. Zeigler, "A Multimodel Methodology for Qualitative Model Engineering," ACM Transactions on Modeling and Computer Simulation, 2 (1), pp. 100, 1992.

Miller V.T., "Heterogeneous Hierarchical Modeling for Knowledge-based Autonomous Systems," Ph.D. Dissertation, Computer and Information Sciences Department, University of Florida, August, 1993.

Praehofer H., "Systems Theoretic Formalisms for Combined Discrete Continuous System Simulation," International Journal of General Systems, 1 9 (3), pp.219-240, 1991.

Zeigler B.P., Multifacetted Modeling and Discrete Event Simulation, Academic Press, London, 1984.

Zeigler B.P., Object Oriented Simulation with Hierarchical, Modular Models, Academic Press, New York, 1990.

AUTHOR BIOGRAPHIES

VICTOR T. MILLER is a visiting assistant professor in the Department of Computer Information and Sciences at the University of Florida. He received a Ph.D. in Computer Engineering from the University of Florida in 1993. His interest and research include simulation, geometric and computational modeling.

PAUL A. FISHWICK is an associate professor in the Department of Computer and Information Sciences at the University of Florida. He received a Ph.D. in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. and at NASA Langley Research Center. His research interests are in computer simulation modeling and analysis methods for complex systems. He is a senior member of the IEEE and the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years and he is on the editorial boards of several journals including the ACM Transactions on Modeling and Computer Simulation, Man and Cybernetics, The Transactions of the Society for Computer Simulation, International Journal of Computer Simulation, and the Journal of Systems Engineering.