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

REPRESENTING AND CONSTRUCTING SYSTEM SPECIFICATIONS USING THE SYSTEM ENTITY STRUCTURE CONCEPTS

Jerzy W. Rozenblit
Bernard P. Zeigler

Dept. of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721

ABSTRACT

A generative approach to constructing system specifications is discussed. This process is intended to support design of hierarchical, multicomponent systems. In previous work, a systems design methodology called Knowledge-Based Simulation Design (KBSD) was developed. KBSD focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. To represent families of design components, a knowledge representation scheme called *system entity structure* (SES) is used. Various modeling formalisms may be used for system specifications in the methodology. Thus, an efficient, generative procedure is needed for constructing specifications for systems represented by an SES. It is shown how system specifications can be managed using a canonical SES representation.

1 INTRODUCTION

Systems theory is a scientific discipline whose primary concern is to provide problem solving methods and tools. The theory owes its utility to the fact that real systems can obey the same "system" laws and show similar patterns of behavior although they are physically very different. This potential isomorphy makes it possible to employ common representations to treat different real systems in a uniform manner (Pichler 1975). Although systems theory has been the subject of intensive studies for a number of years, its problem solving methods have often been ill-understood and ignored by researchers, practicing engineers, and system designers. With the advent of powerful hardware platforms and software development and application tools which implement many of the theory-based concepts, we are observing a renewed interest in the field (Pichler and Schwaertzel 1992).

This paper discusses the systems theory-based support for design of engineering systems. The distinguishing issue in designing such systems is how to endow them with the knowledge required to perform their missions. Conceptual frameworks for intelligent system design (Antsaklis and Passino 1992, Albus 1992) recognize the critical role of models to structure knowledge representation and utilization in such systems. Intelligent system design requires a methodology for task decomposition, assignment of models to subtasks and the integration of models into execution hierarchies matching the task decompositions. Such models can be expressed in various formalisms. However, if the great variety of formalisms is to be marshalled for systems design in this manner, the designer must be able to gain access to them in an organized way.

Computer simulation offers a means to develop and test systems in comparison to real testbed environments (Rozenblit and Zeigler 1990, Zeigler 1990). To support such design, an ideal simulation environment would enable the designer to experiment with a variety of formalisms and models expressed within them. This requires that a design environment accommodate knowledge representation and management schemes as well as formalisms capable of capturing inhomogeneous behaviors of various system components.

2 SYSTEMS THEORY — REVIEW OF BASIC TENETS

Although a generally accepted definition of "system" does not exist, we adopt the following: A *dynamical system* is any formal construct which provides general modeling concepts for various kinds of disciplines (Pichler 1975, Zeigler 1976, Wymore 1976). We distinguish such a mathematical object from any reality that it may represent, using the term *real system* for the latter.

A real system can be represented at varying levels of abstraction. According to the abstraction level,

the system manifests itself in different ways and we use different terms to speak about it. By *system behavior* we denote the way the system appears on its boundary, i.e., how it reacts to inputs by producing outputs. The interior of a system is described by the *system state* and the *system dynamic*. The state represents the condition the system is in at a particular time and the system dynamic governs the way this state changes over time. When the state is represented by one or more variables we speak of *state variables*. The dynamic of the system can be determined by a state transition function which depends on the input and the state itself. How the state and current input appear as output is determined by the *output function*.

When we identify several elements corresponding to parts of the real world, we speak of *system elements* and the state of each of the elements is represented by the system element state. The interdependencies of these elements contributing to the system dynamic is called the *system structure*. When we go further and represent parts of the system by systems themselves and their interdependencies by coupling these parts, we speak of *system components* and *system couplings*. Such a system is called *multicomponent system* or *coupled system*. (In Section 4 we provide formal definitions that reflect the above atomic and coupled system characterizations.)

2.1 Hierarchy of System Specifications

The term model is viewed here as a specification for a system. In general, the term system refers to a description (often mathematical) which captures some of the essential features concerning the system or problem being modelled. Since there are many characteristics of real systems, there are several concepts of the system and thus it is useful to organize the specifications into a coherent whole. In this way we arrive at a stratification of system specifications that starts with intuitive black box concepts at the lower levels and adds more and more constructs for the description of system's internal structure as the levels increase. Klir's epistemological classification of systems description of one such example (Klir 1984, 1985).

Zeigler (1976) provides a hierarchy of system specifications with morphism concepts that enable comparisons between systems specified at any level of abstraction. The hierarchy is defined as follows:

1. System Specification IORO. This type of description is an Input/Output observation relation. It is a classical example of a black box.
2. *System Specification IOFO.* For each input function of a given IORO there exists exactly one output function. This specification is called an I/O function observation.
3. *System Specification S* (often referred to as an I/O system). In addition to input and output sets, a set of states and state-transformation mechanisms have to be defined in this specification.
4. *Structured System Specification.* The specification at this level is the same as the one at level 3 except that each set and function is structured.
5. *System Specification NET (Coupled System).* NET denotes a network of system specifications consisting of a family of systems and a coupling mechanism. This specification is the basis for a hierarchical form of model construction.

The above stratification is independent of any particular modeling formalism. In other words, any formalism may be employed to specify a system at any level. Systems theory affords an integrative view of the diversity of formalisms. Indeed, it regards *formalism* as a modeling language used to define (actually select) a subset of systems. Once a subset is identified, a formalism need express only those features that distinguish a particular system from others in the same subset (Zeigler 1984). In this sense, a system formalism can be regarded as a short-hand means of system specification.

Basic system formalisms are *differential equation* system specifications (DESS), discrete time system specifications (DTSS), and discrete event system specifications (DEVS). The levels of system descriptions and the system formalisms build a crossproduct relation where every combination of system formalism and system level represents a possible modeling concept. The formalisms impose appropriate constraints on the time base, input, output, and state sets, and input, output, and state trajectories. Such constraints circumscribe the systems that can be members of the subset specified by a formalism.

A good review of major modeling formalisms (differential equations, discrete time systems, and discrete event specification) and their associated translation mappings into system specifications are given in (Zeigler 1984 and Praehofer 1991).

3 SYSTEMS DESIGN CONCEPTS

Researches and practitioners continually seek new and better design methods. It has been shown that

design processes are essentially computable (Coyne 1990). Computer-aided design (CAD) became possible for problems in which computational processes were well-defined. First, emphasis was placed on *numerical* type of computation. The advent of symbolic computing, AI, expert systems and knowledge-based systems technologies made it possible for designers to *reason* with knowledge — thus the term *knowledge-based design.*

3.1 Knowledge-Based Simulation Design Methodology

The system design approach proposed by Rozenblit (Rozenblit and Zeigler 1988, 1990, Rozenblit and Huang 1991) termed Knowledge-Based Simulation Design (KBSD), focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. It treats the design process as a series of activities that comprise specification of design levels in a hierarchical manner (decomposition), classification of system components into different variants (specialization), selection of components from specializations and decompositions, development of design models, experimentation and evaluation by simulation, and choice of design solutions.

The design process begins with developing a representation of design components and their variants. A knowledge representation scheme, called *system entity structure* (SES) is used to capture the following three relationships among design elements: *decomposition, taxonomy*, and *coupling*. Decomposition knowledge means that the structure has schemes for representing the manner in which an object (design element) is decomposed into components. Taxonomic knowledge is a representation for the kinds of variants that are possible for an object, i.e., how it can be categorized and subclassified. The synthesis (coupling) constraints impose a manner in which components identified in decompositions can be connected together. The selection constraints limit choices of variants of objects determined by the taxonomic relations.

Beyond this, procedural knowledge is available to select and synthesize the system's components identified in the chosen representation scheme. This selection and synthesis process is called *pruning* (Rozenblit and Huang 1991). Pruning results in a recommendation for a *model composition tree*, i.e., a set of hierarchically arranged design components whose models may reside in a design library.

Performance of alternative design solutions is studied by associating system specifications (in the form of models) with the components of composition trees. Simulations produce dynamic performance data.

3.2 The System Entity Structure

As a step toward a unifying knowledge representation scheme for design support, we have combined the decomposition, taxonomic, and coupling relationships in the *system entity structure* (Zeigler 1984, Rozenblit and Zeigler 1988). In the SES, the representation concerns the admissible variants of components in decompositions and the further specializations of such variants. The interaction of decomposition, coupling and taxonomic relations in an SES affords a compact specification of a family of models for a given domain. In a system entity structure, *entities* refer to conceptual components of reality for which models may reside in a library (model base). Associated with entities are slots for attribute representation. An entity may have several *aspects*, each denoting a decomposition, and therefore having several entities. An entity may also have several *specializations*, each representing a classification of possible variants of the entity.

The construction of, and operations on a system entity structure are governed by a set of axioms, i.e., *uniformity, strict hierarchy, alternating mode, inheritance, valid siblings*, and *attached variables* (Zeigler 1984). The axioms furnish a unifying set of rules for developing and manipulating entity structures.

The system entity structure organizes possibilities for a variety of system decompositions and, consequently, a variety of model constructions. Its generative capability facilitates convenient definition and representation of models and their attributes at multiple levels of aggregation and abstraction. More complete discussions of the system entity structure and its associated structure transformations are presented in (Zeigler 1984, Rozenblit and Zeigler 1988, 1990, Rozenblit and Huang 1991).

Typically, the SES is employed to specify families of design simulation models, generated by pruning a master SES, for a given application domain. Here, we employ SES concepts to provide the knowledge representation structure needed to manage system specification formalisms. First, we provide formal definitions of a system at the atomic and coupled levels. These two definitions are the basis for deriving system specifications in various formalisms (for a set of illustrative examples see (Praehofer 1991)).

4 FORMAL SYSTEM DEFINITIONS

A system (at level 3 of the system specifications hierarchy) is a structure:

$$S =< T, X, \Omega, Q, Y, \delta, \lambda >$$

where:
T is the time base,
X is the input value set,
Ω is the input segment set,
Q is the internal state set,
Y is the output set,
δ: $Q \times \Omega \rightarrow Q$ is the state transition function,
λ: $Q \rightarrow Y$ is the output function.

The input segments of the system S have to be closed with respect to concatenation. In addition, δ must have the semigroup property, i.e., for all $\omega, \omega' \in \Omega$ and for all $q \in Q$ the following equation must hold: $\delta(q, \omega \cdot \omega') = \delta(\delta(q, \omega), \omega')$

The input, state, output sets, and the output function constitute the static structure of the system. The time base, input segment set, and the state transition function are referred to as the system's dynamic structure.

A *Coupling of Systems*, i.e., system network, multicomponent model, is a structure (Zeigler 1984):

$$N =< D, \{S_a\}, \{I_a\}, \{Z_a\} >$$

where:
D is a set of components names,
for each $a \in D$:
S_a is a system, component a,
I_a is a set, the set of influencers of a,
Z_a is a function, the interface mapping of a,

subject to the constraints:

$$S_a =< T, X_a, \Omega_a, Q_a, Y_a, \delta_a, \lambda_a >$$

$$I \subset D$$

$$Z_a\colon \times_{b \in I_a} Y_b \rightarrow X_a$$

This specification designates a set of components $< D, \{S_a\} >$ and a *coupling scheme* $< \{I_a\}, \{Z_a\} >$.

System formalism can be built based on the above generic specifications. To facilitate the construction and management of formalisms, we set up a canonical system entity structure. This SES contains the requisite elements of the formal system's descriptions.

5 SES REPRESENTATION FOR SYSTEM SPECIFICATION FORMALISMS

An SES representation to facilitate management of system specification formalisms is shown in Figure 1.

The root entity of this SES, called the canonical SES, represents a system specification S — either an atomic or a multicomponent one. The constituents of S are sets and functions. Sets can be classified into Input, Output, Time, and State as well as sets of Components or Other, modeler defined objects (e.g., an initial state in a finite state machine specification). Each set is characterized by the type of its elements, e.g., Reals, Integers, etc. In addition, by selecting the System Specification variant from the Set entity, we can include a set of system specifications as elements of the system specification S. This recursive representation affords the construction of formalisms for multicomponent systems. (Note that we must relax the strict hierarchy SES axiom as in Cho (1993) in order to facilitate such a process.)

The functions are: Transition, Output, and Segment, Coupling (I/O Translation), or Other (i.e., modeler defined functions, for example, a rate of change, or time advance function). Each function has an attribute type which characterizes its properties, e.g., step, piecewise continuous, piecewise differentiable.

In system design, the canonical SES of system specifications is used in conjunction with an SES for an application domain. Pruning the application SES generates a composition tree for the system model specification. The model composition tree is a tree whose leaf nodes are system specifications. These are atomic components which are coupled in a hierarchical manner. In the next section, we show how the atomic and coupled level specifications can constructed based on the composition tree and canonical SES representations.

As an example, Figure 2 depicts a composition tree of a two component system. Assume, that the two subsystems S_1 and S_2 are connected in series as shown in Figure 3.

The resultant is the system S whose formal specification should now be derived. Let us illustrate this process at both the atomic and the coupled system level.

5.1 Atomic System Specification

To generate a formal specification for the atomic components S_1 and S_2, design constraints and requirements as well as physical characteristics of the model's counterpart (i.e., real system) are analyzed. This is done in order to determine appropriate types of sets and functions needed to characterize the dynamic behavior of the components under consideration.

Assume that both subsystems exhibit an inherently discrete behavior. The modeler may prune the canonical SES so that the DEVS formalism is selected for Formalism-type. Since DEVS is a formalism that will

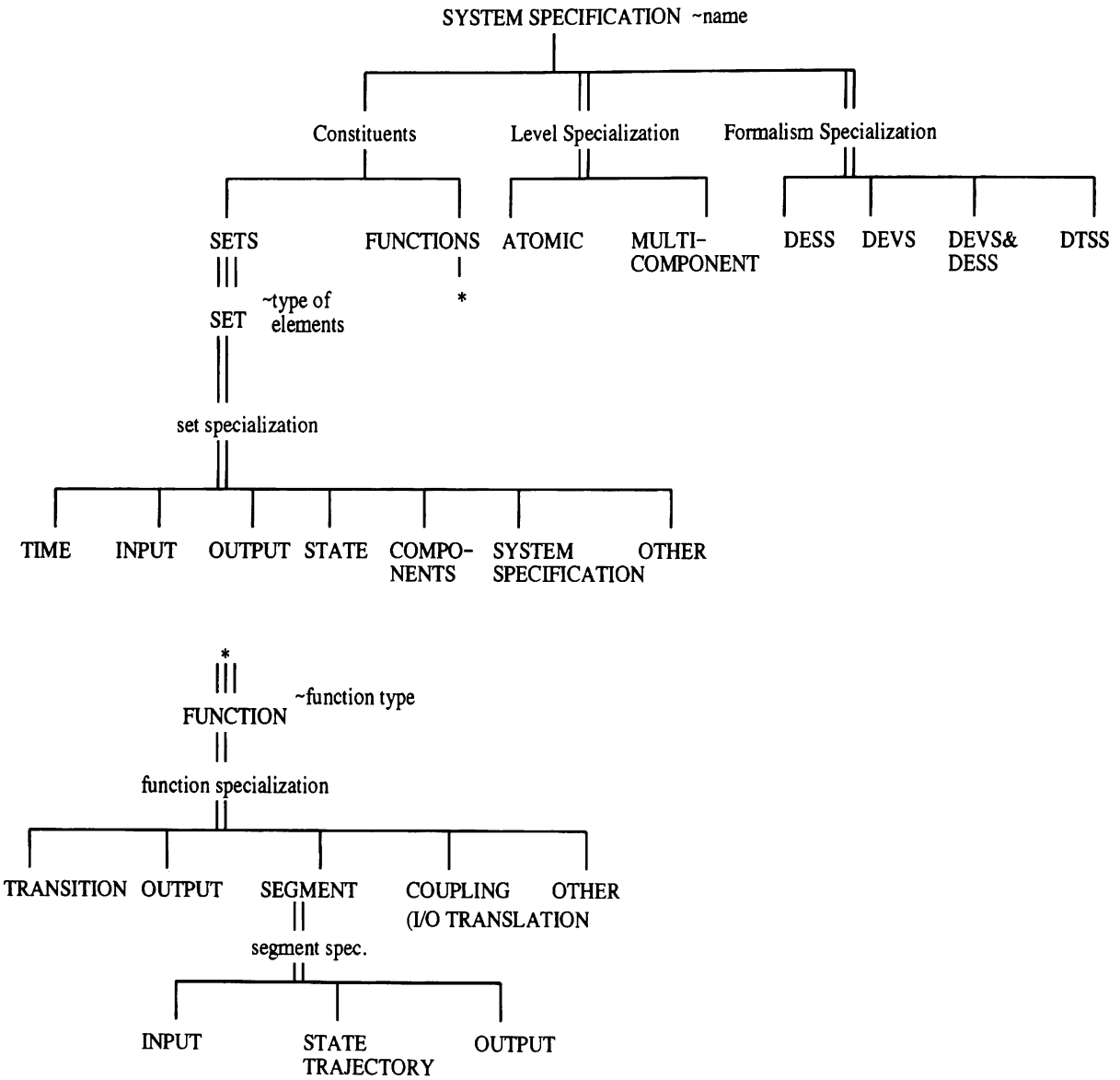

Figure 1: Canonical SES of System Specification Elements

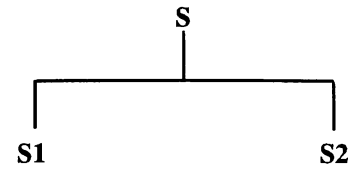

Figure 2: Simple Composition Tree

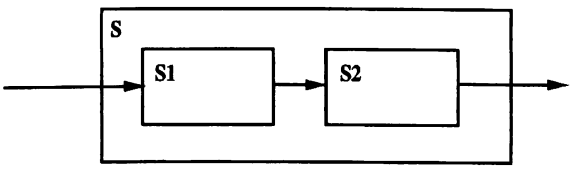

Figure 3: System S as a Series Coupling of Subsystems S1 and S2

be known to the system, its constituent slots will be automatically retrieved leaving only their values to be supplied by the modeler. Such a selection might be presented in the form of a frame data structure as follows:

Pruned Canonical SES for an Atomic Specification (DEVS)

FRAME S_1
System Specification Type: Atomic
Constituents: Sets, Functions
Sets:
Time:
Type of Elements: Reals
Input: X
Type of Elements: Discrete Events
Output: Y
Type of Elements: Reals
State: S
Type of Elements:
Nonnegative Integers
Components: Null
System Specification: Null
Other: Null
Functions:
Transition:
Internal: δ_ϕ: $S \rightarrow S$
External: δ_{ex}: $Q \times X \rightarrow S$
where Q is given by:
$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$
Output: λ: $S \rightarrow Y$
Other: Time advance function –
ta: $S \rightarrow R^+_{0,\infty}$
Coupling (Input-to-Output Translation):
None
Segment: None

Choosing a DEVS atomic model for the component S_2, a Frame S_2 is developed in a manner similar to that of Frame S_1. We now proceed to illustrate how to construct a coupled level (network) system specification.

5.2 Coupled System Specification

The specification of the system S can be obtained by coupling the specifications of its components. Since the DEVS formalism is closed under coupling, the modeler may prune the canonical SES to select the DEVS formalism for the multicomponent level specialization.

Pruned Canonical SES for a Multicomponent Specification (DEVS)

FRAME S
System Specification Type: Multicomponent Discrete Event Network
Constituents: Sets, Functions
Sets:
Components:
D - component names $\{S_1, S_2\}$
I - influencees: $\{S_1 : \{\}, S_2 : \{S_1\}\}$
System Specifications:
Frame S_1
Frame S_2
Functions:
Coupling: (I/O) Translation):
Z_{12} : $Y_1 \rightarrow X_2$
defined as the identity mapping
Other:
Select: $2^D \rightarrow D$
defined by linear order $< 1, 2 >$

Note that in environments such as DEVS-Scheme (Zeigler 1990) and STIMS (Pichler and Schwaertzel 1992), the slots in the frames can be filled in a manner that is user-oriented, yet very close to their "pure" mathematical forms. Note also that constraints (e.g., no direct feedbacks in a DEVS network) must be added to the SES to ensure the compatibility of component formalisms with the coupled system formalism at the next higher level .

6 OBJECT ORIENTED IMPLEMENTATION OF THE CANONICAL SES

The implementation of the canonical SES can be supported by the object-oriented programming

paradigm. In a realization of systems theory instrumented design framework, we must represent the following:

- the system formalisms,
- the constituents of a formalism, viz., the sets and functions of the formalism,
- the formalism specialization hierarchy,
- the constraints and restrictions defined for the formalism and its constituents,
- the dynamic characteristics of the formalisms, viz., its simulation algorithm in a multiformalism framework, and
- the operations defined for the formalisms.

In (Zeigler et al. 1993), we discuss the canonical SES implementation principles in detail. Here, we summarize the basic tenets of our approach.

Mapping the system formalisms and models to an object-oriented is accomplished as follows: the formalisms can be represented by class definitions; the models of the different formalisms are realized by instances of the particular formalism classes.

The constituents of the models are defined by slot (also called instance variable) definitions. Operations on these slots in the different formalism classes can be unified by the generic function concept of CLOS (Keene 1989).

To represent the specialization relationship of the different formalisms, the object oriented paradigm offers the concept of class hierarchy with multiple inheritance. However, due to the diversity of the formalisms and their great variety of interrelations, such a representation proves to be difficult. Mittelmann and Praehofer (1991) propose a knowledge representation scheme where the constituents of system formalisms are implemented by abstract classes whereas the actual formalisms are defined through dynamic class definitions multiply inheriting from the appropriate constituent classes. They also provide an algorithm to set up an inheritance hierarchy which minimizes the duplication of slots and methods in a complex specialization hierarchy, an objective of any object oriented implementation.

Methods can also be used to implement the constraints defined on the constituents of system formalisms. We outline this in detail in (Zeigler et al. 1993).

7 CLOSING REMARKS

We have discussed the utility of adopting a formal knowledge representation scheme for organizing and managing generic classes of systems formalisms within a design framework. Especially when applied to intelligent system design, the number of formalisms being proposed by researchers is growing tremendously. Our approach is intended to make such formalisms attractive to researchers, practicing designers, and systems engineers by providing concepts and tools to manage and integrate them.

REFERENCES

Albus, J.S. 1992. A Reference Model Architecture for Intelligent Systems Design. In *An Introduction to Intelligent and Autonomous Control.* eds. Antsaklis, P. J. and K. M. Passino. Kluwer Academic Publishers.

Antsaklis, P.J., and K.M. Passino. 1992. Introduction to Intelligent Control Systems with High Degrees of Autonomy. In *An Introduction to Intelligent and Autonomous Control.* eds. Antsaklis, P. J. and K. M. Passino. Kluwer Academic Publishers.

Cho, T. 1993. A Hierarchical, Modular Simulation Environment for Flexible Manufacturing, PhD Dissertation, Dept. of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona.

Coyne, R. D. et al. 1990. *Knowledge-Based Design Systems.* Addison-Wesley.

Keene, S. E. 1989. *Object-Oriented Programming in Common LISP.* Addison Wesley.

Klir, G. J. 1984. General Systems Framework for Inductive Modeling. In *Simulation and Model-Based Methodologies: an Integrative View*, eds. Oren, T.I. et al. Springer-Verlag. 69-90.

Klir, G. J. 1985. *Architectures of System Problem Solving.* Plenum Press.

Mittelmann, R., and H. Praehofer. 1990. Design of an Object Oriented Kernel System for Computer Aided Systems Theory and Systems Theory Instrumented Modelling and Simulation. In *Computer Aided Systems Theory — EUROCAST '89*, eds. F. Pichler and R. Moreno-Diaz, 76-85. Springer-Verlag.

Pichler, F. 1975. Mathematische Systemtheorie. Walter de Gruyter.

Pichler, F. 1984. Symbolic Manipulation of System Models. In *Simulation and Model-Based Methodologies: an Integrative View.* eds. Oren, T.I. et. al. Springer-Verlag. 217-234.

Pichler, F., and H. Schwaertzel. 1992. (Eds.) *CAST Methods in Modeling: Computer Assisted Systems Theory for the Design of Intelligent Systems.* Springer-Verlag.

Praehofer, H. 1991. Systems Theoretic Foundations for Combined Discrete Continuous System Simulation. PhD Dissertation, Department of Systems Theory, University of Linz, Austria.

Rozenblit, J. W., and B.P. Zeigler. 1988. Design and Modeling Concepts. In *International Encyclopedia of Robotics, Applications and Automation*, ed. Dorf, R. John Wiley and Sons. 308-322.

Rozenblit, J. W., and B.P. Zeigler. 1990. Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Application, *Transactions of the Society for Computer Simulation*, 7:3: 195-228.

Rozenblit, J. W., and Y. M. Huang. 1991. Rule-Based Generation of Model Structures in Multifacetted Modeling and System Design. *ORSA Journal on Computing*. 3:4: 330-344.

Wymore, W. A. 1976. *Systems Engineering Methodology for Interdisciplinary Teams*, John Wiley and Sons, New York.

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

Zeigler, B. P. 1984. *Multifaceted Modelling and Discrete Event Simulation*, Academic Press.

Zeigler, B. P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press.

Zeigler, B. P., H. Praehofer, and J. W. Rozenblit. 1993. Integrating Systems Formalisms: How Object-Oriented Programming Supports Cast for Intelligent Systems Design. *Journal of Systems Engineering.* (in press).

AUTHOR BIOGRAPHIES

JERZY ROZENBLIT is an Associate Professor of Electrical and Computer Engineering at The University of Arizona. His research interests lie in the application of artificial intelligence and simulation modeling to systems design. He is a member of IEEE, ACM, and SCS.

BERNARD ZEIGLER is a Professor of Electrical and Computer Engineering at The University of Arizona. He is the author of *Theory of Modelling and Simulation*, *Multifaceted Modelling and Discrete Event Simulation*, and *Object-Oriented Simulation with Hierarchical, Modular Models.*