Zeigler, B. P., Muzy, A., & Kofman, E. (2018). Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations. Academic Press.

Book · December 2018

DOI: 10.1016/C2016-0-03987-6

3 authors:

Bernard Phillip Zeigler
The University of Arizona

Alexandre Muzy
Université Côte d'Azur, CNRS

Ernesto Kofman
National Scientific and Technical Research Council

AUTHOR QUERY FORM

ELSEVIER	**Book:** Theory of Modeling and Simulation **Chapter:** 01	**Please e-mail your responses and any corrections to:** **E-mail:** **Ni.Kumar@elsevier.com**

Dear Author,

Please check your proof carefully and mark all corrections at the appropriate place in the proof. **It is crucial that you NOT make direct edits to the PDF using the editing tools as doing so could lead us to overlook your desired changes.** Rather, please request corrections by using the tools in the Comment pane to annotate the PDF and call out the changes you would like to see. To ensure fast publication of your paper please return your corrections within 48 hours.

For correction or revision of any artwork, please consult http://www.elsevier.com/artworkinstructions

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof.

Location in chapter	Query / Remark: Click on the Q link to find the query's location in text Please insert your reply or correction at the corresponding line in the proof
Q1	An extra opening parenthesis before "and UML can be used" has been deleted. Please check, and correct if necessary. (p. 9/ line 35) Please check this box or indicate your approval if you have no corrections to make to the PDF file ☐

Thank you for your assistance.

CHAPTER

INTRODUCTION TO SYSTEMS MODELING CONCEPTS 1

CONTENTS

This chapter introduces some key concepts that underlie the framework and methodology for modeling and simulation (M&S) originally presented in "Theory of Modeling and Simulation" published in 1976 – referred to hereafter as TMS76 to distinguish it from the current revised third edition TMS2018. Perhaps the most basic concept is that of mathematical systems theory. First developed in the nineteen sixties, this theory provides a fundamental, rigorous mathematical formalism for representing dynamical systems. There are two main, and orthogonal, aspects to the theory:

- *Levels of system specification*: these are the levels at which we can describe how systems behave and the mechanisms that make them work the way they do.
- *Systems specification formalisms*: these are the types of modeling styles, such continuous or discrete, that modelers can use to build system models.

Although the theory is quite intuitive, it does present an abstract way of thinking about the world that you will probably find unfamiliar. So we introduce the concepts in a spiral development consisting of easy-to-grasp stages – with each spiral revolution returning to a more faithful version of the full story.

Theory of Modeling and Simulation. https://doi.org/10.1016/B978-0-12-813370-5.00009-2

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 1

In this chapter we first introduce some basic systems concepts, then motivate the systems specification formalisms by describing their evolution over time. This also provides a way to point out the differences between earlier editions and this one (TMS2018). Finally, we discuss the levels of system specification, illustrating them with familiar examples. In this ground stage of our spiral development, the presentation is informal and prepares the way for the framework for M&S that comes in the next chapter. Later, in the second part of the book, we return to a more rigorous development of the concepts to lay a sound basis for the developments to come in the third part.

1.1 SYSTEMS SPECIFICATION FORMALISMS

System theory distinguishes between system structure (the inner constitution of a system) and behavior (its outer manifestation). Viewed as a black box (Fig. 1.1) the external behavior of a system is the relationship it imposes between its input time histories and output time histories. The system's input/output behavior consists of the pairs of data records (input time segments paired with output time segments) gathered from a real system or model. The internal structure of a system includes its state and state transition mechanism (dictating how inputs transform current states into successor states) as well as the state-to-output mapping. Knowing the system structure allows us to deduce (analyze, simulate) its behavior. Usually, the other direction (inferring structure from behavior) is not univalent – indeed, discovering a valid representation of an observed behavior is one of the key concerns of the M&S enterprise.

An important structure concept is that of *decomposition* namely, how a system may be broken down into component systems (Fig. 1.2). A second concept is that of *composition*, i.e., how component systems may be coupled together to form a larger system. Systems theory is *closed under composition* in that the structure and behavior of a composition of systems can be expressed in the original system theory terms. The ability to continue to compose larger and larger systems from previously constructed components leads to hierarchical construction. Closure under composition guarantees that such a composition results in a system, called its *resultant*, with well-defined structure and behavior. *Modular*

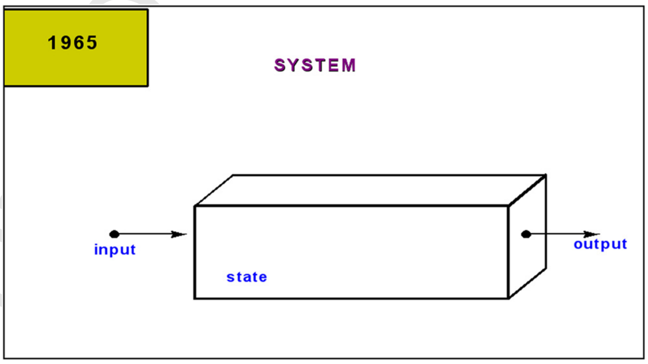

FIGURE 1.1

Basic System Concepts.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 2

B978-0-12-813370-5.00009-2, 00001

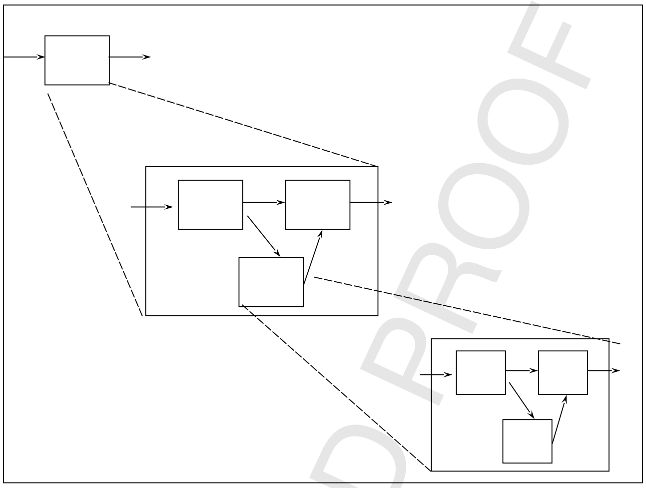

FIGURE 1.2

Hierarchical System Decomposition.

systems have recognized input and output ports through which all interaction with the environment occurs. They can be coupled together by coupling output ports to input ports and can have hierarchical structure in which component systems are coupled together to form larger ones.

The difference between a decomposed systems, as in Fig. 1.2, and undecomposed systems, as in Fig. 1.1, provides our first introduction to levels of systems specification. We'll say later that the former are at a higher level of specification than the latter since they provide more information about the structure of the system.

1.1.1 RELATION TO OBJECT ORIENTATION

Models developed in a system theory paradigm bear a resemblance to concepts of object-oriented programming. Both objects and system models share a concept of internal state. However, mathematical systems are formal structures that operate on a time base while programming objects typically do not have an associated temporal semantics. Objects in typical object oriented paradigms are not hierarchical or modular in the sense just described. The coupling concept in modular systems provides a level of delayed binding – a system model can place a value on one of its ports but the actual destination of this output is not determined until the model becomes a component in a larger system and a coupling scheme is specified. It can therefore: a) be developed and tested as a stand alone unit, b) be placed in a model repository and reactivated at will and c) reused in any applications context in which its behavior is appropriate and coupling to other components makes sense.

While coupling establishes output-to-input pathways, the systems modeler is completely free to specify how data flows along such channels. Information flow is one of many interactions that may be represented. Other interactions include physical forces and fields, material flows, monetary flows, and social transactions. The systems concept is broad enough to include the representation of any of these

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 3

B978-0-12-813370-5.00009-2, 00001

and supports the development of M&S environments that can make including many within the same large-scale model.

Although systems models have formal temporal and coupling features not shared by conventional objects, object-orientation does provide a supporting computational mechanism for system modeling. Indeed, there have been many object-oriented implementations of hierarchical, modular modeling systems These demonstrate that object-oriented paradigms, particularly for distributed computing, can serve as a strong foundation to implement the modular systems paradigm.

1.1.2 EVOLUTION OF SYSTEMS FORMALISMS

As in many situations, portraying the evolution of an idea may help in the understanding of the complexities as they develop. Fig. 1.3 depicts the basic systems modeling formalisms as they were presented in the first edition, TMS76. This edition was the first book to formulate approaches to modeling as system specification formalisms – shorthand means of delineating a particular system within a subclass of all systems. The traditional differential equation systems, having continuous states and continuous time, were formulated as the class of DESS (Differential Equation System Specifications). Also, systems that operated on a discrete time base such as automata were formulated as the class of DTSS (Discrete Time System Specifications). In each of these cases, mathematical representation had proceeded their computerized incarnations (it has been three hundred years since Newton-Leibnitz!).

However, the reverse was true for the third class, the Discrete Event System Specifications (DEVS). Discrete event models were largely prisoners of their simulation language implementations or algorithmic code expressions. Indeed, there was a prevalent belief that discrete event "world views" constituted new mutant forms of simulation, unrelated to the traditional mainstream paradigms. Fortunately, that situation has begun to change as the benefits of abstractions in control and design became clear. Witness the variety of discrete event dynamic system formalisms that have emerged (Ho, 1992). Examples are Petri Nets, Min-Max algebra, and GSMP (generalized semi-Markov processes). While each one has its

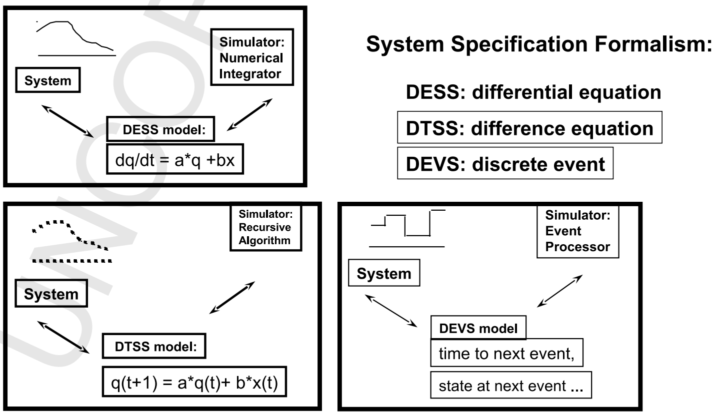

FIGURE 1.3

Basic Systems Specification Formalisms.

ZEIGLER, 978-0-12-813370-5

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 4

B978-0-12-813370-5.00009-2, 00001

application area, none were developed deliberately as subclasses of the systems theory formalism. Thus to include such a formalism into an organized system-theory based framework requires "embedding" it into DEVS.

"Embedding." What could such a concept mean? The arrows in Fig. 1.4 indicate subclass relationships; for example, they suggest that DTSS is a "subclass of" DEVS. However, it is not literally true that any discrete time system is also discrete event system (their time bases are distinct, for example). So we need a concept of simulation that allows us to say when one system can do the essential work of another. One formalism can be embedded in another if any system in the first can be simulated by some system in the second. Actually, more than one such relationship, or morphism, may be useful, since, as already mentioned, there are various levels of structure and behavior at which equivalence of systems could be required. As a case in point, the TMS76 edition established that any DTSS could be simulated by a DEVS by constraining the time advance to be constant. However, this is not as useful as it could be until we can see how it applies to decomposed systems. Until that is true, we either must reconstitute a decomposed discrete time system to its resultant before representing it as a DEVS or we can represent each DTSS component as a DEVS but we can't network the DEVS together to simulate the resultant. TMS2000 established this stronger simulation relation and we discuss its application in Chapters 18 and 20 of this edition.

1.1.3 CONTINUOUS AND DISCRETE FORMALISMS

Skipping many years of accumulating developments, the next major advance in systems formalisms was the combination of discrete event and differential equation formalisms into one, the DEV&DESS. As shown in Fig. 1.5, this formalism subsumes both the DESS and the DEVS (hence also the DTSS) and thus supports the development of coupled systems whose components are expressed in any of the basic formalisms. Such *multi-formalism* modeling capability is important since the world does not

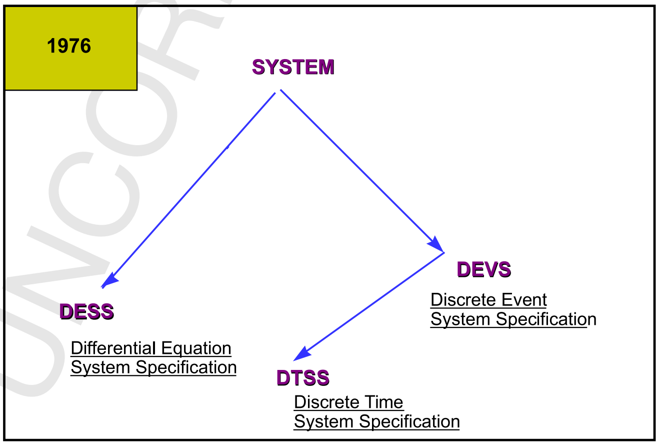

FIGURE 1.4

The Dynamics of Basic System Classes.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 5

B978-0-12-813370-5.00009-2, 00001

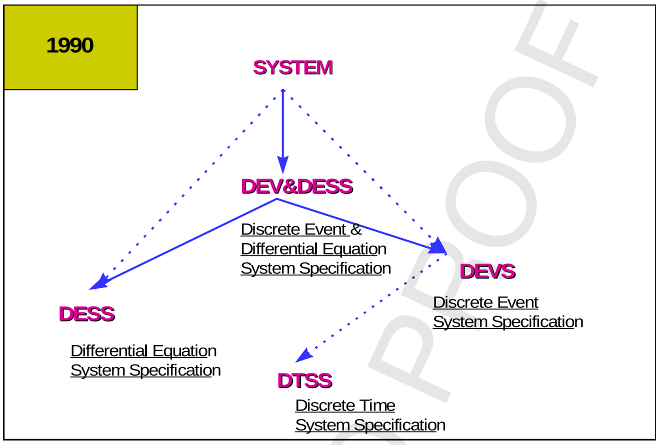

FIGURE 1.5

Introducing the DEV&DESS Formalism.

usually lend itself to using one form of abstraction at a time. For example, a chemical plant is usually modeled with differential equations while its control logic is best designed with discrete event formalisms. In 1990, Praehofer (Praehofer, 1991) showed that DEV&DESS was closed under coupling and in order to do so, had to deal with the pairs of input-output interfaces between the different types of systems. Closure under coupling also required that the DEV&DESS formalism provide a means to specify components with intermingled discrete and continuous expressions. Finally, simulator algorithms (so called *abstract simulator*) had to be provided to establish that the new formalism could be implemented in computational form (look ahead to Chapter 9 to see how this was all accomplished).

1.1.4 QUANTIZED SYSTEMS

TMS2000 built on the advances since 1976 especially in the directions pointed to by the introduction of DEV&DESS. Since parallel and distributed simulation has become a dominant form of model execution, and discrete event concepts best fit with this technology, the focus turned to a concept called the DEVS *bus*. This concept, introduced in 1996, concerns the use of DEVS models, as a "wrappers" to enable a variety of models, to interoperate in a networked simulation. It was particularly germane to the High Level Architecture (HLA) defined by the United States Department of Defense. One way of looking at this idea is that we want to embed any formalism, including for example, the DEV&DESS, into DEVS. Another way was to introduce a new class of systems, called the Quantized System, as illustrated in Fig. 1.6. In such systems, both the input and output are quantized. As an example, an analog-to-digital converter does such quantization by mapping a real number into a finite string of digits. In general, quantization forms equivalence classes of outputs that then become indistinguishable for downstream input receivers, requiring less data network bandwidth, but also possibly incurring error.

Quantization provides a process for representing and simulating continuous systems that is an alternative to the more conventional discretization of the time axis. While discretization leads to discrete

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 6

B978-0-12-813370-5.00009-2, 00001

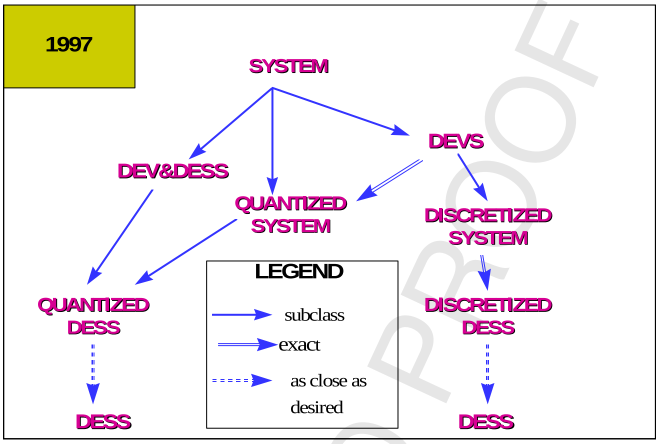

FIGURE 1.6

Introducing Quantized Systems.

time systems, quantization leads to discrete event systems. The theory of quantized state systems that has been developed since 2000 is presented in Chapter 20 of this edition.

When we restrict quantization to differential equation systems, we can express the resulting class, Quantized DESS, within DEV&DESS and study its properties, especially from the point of view of the DEVS bus. We can then study the approximation capability and simulation efficiency of DEVS in distributed simulation in comparison with classical time stepped integration and simulation approaches. Particularly with respect to reduction of message passing and network bandwidth (a major concern in distributed simulation) promising results are being obtained.

1.1.5 EXTENSIONS OF DEVS

Various extensions of DEVS have been developed as illustrated in Fig. 1.7. In the interest of space conservation, some of them are not discussed here while still available in TMS2000. Since our focus here is on Iterative System Specification as a modeling formalism, we present new types of such models in Chapter 12. These developments expand the classes of system models that can be represented and integrated within both DEVS and the parent systems theory formalism and UML can be used to classify newly developed variants and extensions (Blas and Zeigler, 2018).

These developments lend credence to the claim that DEVS is a promising computational basis for analysis and design of systems, particularly when simulation is the ultimate environment for development and testing (Fig. 1.8). The claim rests on the *universality* of the DEVS representation, namely the ability of DEVS bus to support the basic system formalisms. TMS2000 went some distance toward substantiating the claim that DEVS is the unique form of representation that underlies any system with discrete event behavior. In this edition, we expand the scope to consider Iterative Specification of Systems and the DEVS Bus support of co-simulation, the ability to correctly interoperate simulators embedding models from diverse formalisms within an integrated distributed simulation environment.

ZEIGLER, 978-0-12-813370-5

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (Irinkeviciute) Prn: 2018/06/22;12:57:07 p. 7

B978-0-12-813370-5.00009-2, 00001

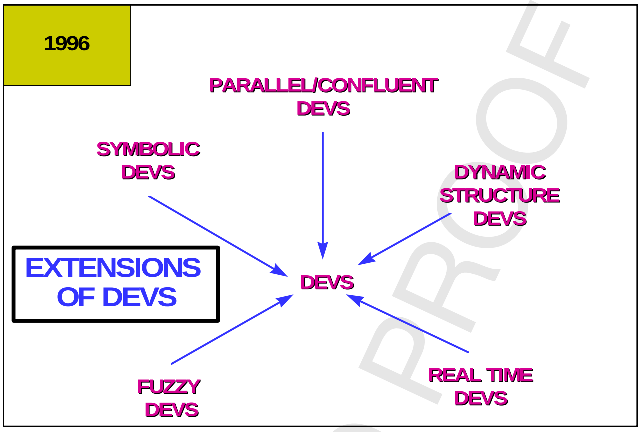

FIGURE 1.7

Extensions of the DEVS Formalism.

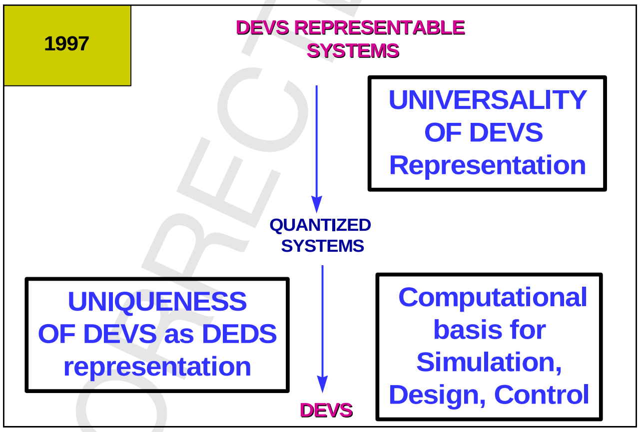

FIGURE 1.8

DEVS as a Computational Basis for Simulation, Design and Control.

1.2 LEVELS OF SYSTEM KNOWLEDGE

As already mentioned, the systems specification hierarchy is the basis for a framework for M&S which sets forth the fundamental entities and relationships in the M&S enterprise. The hierarchy is first presented in an informal manner and later in Chapter 5 in its full mathematical rigor. Our presentation starts with a review of George Klir's (Klir, 1985) systems framework.

Table 1.1 identifies four basic levels of knowledge about a system recognized by Klir. At each level we know some important things about a system that we didn't know at lower levels. At the lowest level,

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:07 p. 8

B978-0-12-813370-5.00009-2, 00001

Table 1.1 Levels of System Knowledge

Level	Name	What we know at this level
0	Source	what variables to measure and how to observe them
1	Data	data collected from a source system
2	Generative	means to generate data in a data system
3	Structure	components (at lower levels) coupled together to form a generative system

the *source level* identifies a portion of the real world that we wish to model and the means by which we are going to observe it. As the next level, the data level is a data base of measurements and observations made for the source system. When we get to Level 2, we have the ability to recreate this data using a more compact representation, such as a formula. Since typically, there are many formulas or other means to generate the same data, the generative level, or particular means or formula we have settled on, constitutes knowledge we didn't have at the data system level. When people talk about models in the context of simulation studies they are usually referring to the concepts identified at this level. That is, to them a model means a program to generate data. At the last level, the structure level, we have a very specific kind of generative system. In other words, we know how to generate the data observed at Level 1 in a more specific manner – in terms of component systems that are interconnected together and whose interaction accounts for the observations made. When people talk about systems, they are often referring to this level of knowledge. They think of reality as being made up of interacting parts – so that the whole is the sum (or a sometimes claimed, more, or less, than the sum) of its parts. Although some people use the term 'subsystems' for these parts, we call them **component** systems (and reserve the term subsystem for another meaning).

As we have suggested, Klir's terms are by no means universally known, understood, or accepted in the M&S community. However, his framework is a useful starting point since it provides a unified perspective on what are usually considered to be distinct concepts. From this perspective, there are only three basic kinds of problems dealing with systems and they involve moving between the levels of system knowledge (Table 1.2). In *systems analysis*, we are trying to understand the behavior of an existing or hypothetical system based on its known structure. *Systems inference* is done when we don't know what this structure is – so we try to guess this structure from observations that we can make. Finally, in *systems design*, we are investigating the alternative structures for a completely new system or the redesign of an existing one.

The central idea is that when we move to a lower level, we don't generate any really new knowledge – we are only making explicit what is implicit in the description we already have. One could argue that making something explicit can lead to insight, or understanding, which is a form of new knowledge, but Klir is not considering this kind of subjective (or modeler dependent) knowledge. In the M&S context, one major form of *systems analysis* is computer simulation which generates data under the instructions provided by a model. While no new knowledge (in Klir's sense) is generated, interesting properties may come to light of which we were not aware before the analysis. On the other hand, *systems inference* and *systems design* are problems that involve **climbing up** the levels. In both cases we have a low level system description and wish to come up with an equivalent higher level one. For *systems inference*, the lower level system is typically at the data system level, being data that we have observed from some existing source system. We are trying to find a generative system, or even a structure system, that can recreate the observed data. In the M&S context, this is usually called *model*

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 9

B978-0-12-813370-5.00009-2, 00001

Table 1.2 Fundamental Systems Problems

Systems Problem	Does source of the data exist? What are we trying to learn about it?	Which level transition is involved?
systems analysis	The system being analyzed may exist or may be planned. In either case we are trying to understand its behavioral characteristics.	moving from higher to lower levels, e.g., using generative information to generate the data in a data system
systems inference	The system exists. We are trying to infer how it works from observations of its behavior.	moving from lower to higher levels, e.g., having data, finding a means to generate it
systems design	The system being designed does not yet exist in the form that is being contemplated.	We are trying to come up with a good design for it. moving from lower to higher levels, e.g. having a means to generate observed data, synthesizing it with components taken off the shelf.

construction. In the case of systems design, the source system typically does not yet exist and our objective is to build one that has a desired functionality. By functionality we mean what we want the system to do; typically, we want to come up with a structure system, whose components are technological, i.e., can be obtained off-the-shelf, or built from scratch from existing technologies. When these components are interconnected, as specified by a structure system's coupling relation, the result should be a real system that behaves as desired.

It is interesting to note that the process called *reverse engineering* has elements of both inference and design. To reverse engineer an existing system, such as was done in the case of the cloning of IBM compatible PCs, an extensive set of observations is first made. From these observations, the behavior of the system is inferred and an alternative structure to realize this behavior is designed thus bypassing patent rights to the original system design!

1.3 INTRODUCTION TO THE HIERARCHY OF SYSTEMS SPECIFICATIONS

At about the same time (in the early 1970's) that Klir introduced his epistemological (knowledge) levels, TMS76 formulated a similar hierarchy that is more oriented toward the M&S context. This framework employs a general concept of *dynamical system* and identifies useful ways in which such a system can be specified. These ways of describing a system can be ordered in levels as in Table 1.3. Just as in Klir's framework, at each level more information is provided in the specification that cannot be derived from lower levels. As can be seen in Table 1.3, these levels roughly correspond to those of Klir's framework.

The major difference between the two frameworks is that the System Specification Hierarchy recognize that simulation deals with **dynamics**, the way in which systems behave over time. Therefore, time is the base upon which all events are ordered. We also view systems as having *input* and *output* interfaces through which they can interact with other systems. As illustrated in Fig. 1.9, systems receive stimuli ordered in time through their *input ports*, and respond on their *output ports*. The term "port" sig-

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 10

B978-0-12-813370-5.00009-2, 00001

Table 1.3 Relation between System Specification Hierarchy and Klir's levels

Level	Specification Name	Corresponds to Klir's	What we know at this level
0	Observation Frame	Source System	how to stimulate the system with inputs; what variables to measure and how to observe them over a time base;
1	I/O Behavior	Data System	time-indexed data collected from a source system; consists of input/output pairs.
2	I/O Function		Knowledge of initial state; given an initial state, every input stimulus produces a unique output.
3	State Transition	Generative System	How states are affected by inputs; given a state and an input what is the state after the input stimulus is over; what output event is generated by a state.
4	Coupled Component	Structure System	Components and how they are coupled together. The components can be specified at lower levels or can even be structure systems themselves – leading to hierarchical structure.

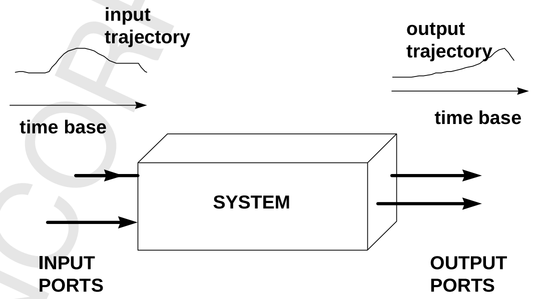

FIGURE 1.9

Input/Output System.

nifies a specific means of interacting with the system. Whether by stimulating it (input) or observing it (output). The time-indexed inputs to systems are called *input trajectories*; likewise, their time-indexed outputs are called *output trajectories*. Ports are the only channels through which one can interact with the system. This means that system are *modular*. While Klir's framework can include dynamics, in-

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 11

put/output ports and modularity, it is not dedicated to these concepts. However, understanding these concepts is critical to effectively solving the problems that arise in M&S.

Before discussing each level of the specification hierarchy in some detail, let's observe that we could have the very same real world object specified simultaneously at each of the levels. Thus there should be a way to associate the next lower level specification with any given one. This association concept is illustrated in Fig. 1.10. For example, if we have know the detailed structure at the Coupled Component level, then we ought to be able to construct the corresponding specification at the State Transition level. The hierarchy is set up to provide such an *association mapping* at each (other than the lowest) level. Indeed, this is the formal version of climbing down the levels just discussed. Since the association mapping is not necessarily one to one, many upper level specifications may map to the same lower level one. This is the underlying reason why climbing up the levels is much harder than climbing down the levels. Indeed, when we select one of the associated upper level specifications for a given lower level one, we are gaining knowledge we didn't have at the lower level.

1.4 THE SPECIFICATION LEVELS INFORMALLY PRESENTED

1.4.1 OBSERVATION FRAME

The Observation Frame specifies how to stimulate the system with inputs; what variables to measure and how to observe them over a time base.

As an example, Fig. 1.11 shows a forest subject to *lightning*, *rain* and *wind*, modeled as input ports and *smoke* produced from fire, represented as an output port. This is a level 0 or Observation Frame specification. Note the choice of variables we included as ports and their *orientation* (i.e., whether they

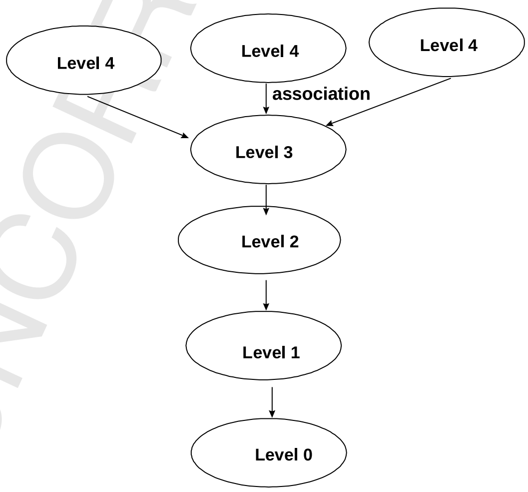

FIGURE 1.10

Association between levels of the System Specification Hierarchy.

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (Irinkeviciute) Prn: 2018/06/22;12:57:08 p. 12

B978-0-12-813370-5.00009-2, 00001

FIGURE 1.11

A forest specified as a system in the Observation Frame (Level 0).

are input or output ports). We could have chosen differently. For example, we could have included an output port to represent heat radiation. Moreover, rather than representing each variable by a single value, it can be distributed over space, i.e., represented by an array of values. Such choices depend on our modeling objectives and are specified through *experimental frames*, a concept which we discuss in Chapter 2.

Fig. 1.12 shows some examples of input and output trajectories. The input trajectory on the *lightning* port shows a bolt occurring at some particular time t0. Only one such bolt occurs in the time period shown. The *smoke* output trajectory, at the top, depicts a gradual build up of *smoke* starting at t0 (so presumably, caused by a fire started by the lightning bolt). The possible values taken on by *smoke*, called its range, would result from some appropriate measurement scheme, e.g., measuring density of particulate material in grams/cubic meter. The pair of input, and associated output, trajectories is called a *input/output* (or I/O) pair. Fig. 1.12 also displays a second I/O pair with the same input trajectory but different output trajectory. It represents the fact that there may be many responses to the same stimulus. In the second case, lightning did not cause a major fire, since the one that broke out quickly died. Such multiple output trajectories (for the same input trajectory) are characteristic of knowledge at Level 1. Knowing how to disambiguate these output trajectories is knowledge we will gain at the next level.

1.4.2 I/O BEHAVIOR AND I/O FUNCTION

The collection of all I/O pairs gathered by observation is called the *I/O Behavior* of a system. Returning to Table 1.3, this represents a system specification at Level 1. Now suppose that we are able to uniquely

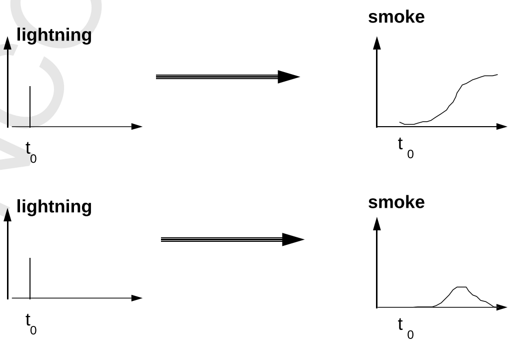

FIGURE 1.12

Some Input-Output Pairs for the Forest System Frame of Fig. 1.11.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 13

predict the response of the smoke output to a lightning bolt. For example, suppose we know that if the vegetation is dry, then a major fire will ignite, but if the vegetation is moist then any fire will quickly die. Having such a factor represents knowledge at Level 2, that of the *I/O Function*. Here, in addition to lower level information, we add *initial states* to the specification – when the initial state is known, there is a functional relationship between input and output trajectories, i.e., the initial state determines the **unique** response to any input (Fig. 1.13A).

1.4.3 STATE TRANSITION SYSTEM SPECIFICATION

At the next level (3) of system specification, we can specify not only initial state information but also how the *state changes* as the system responds to its input trajectory. Fig. 1.13B and C illustrate this important concept. Fig. 1.13B presents the situation where the forest is in state (dry vegetation, unburned) when a lightning bolt occurs at time *t0*. The state that the forest is in at time *t1* when a second bolt occurs is (dry vegetation, burnt) reflecting the fact that a fire has ignited. Since the forest is in a different state, the effect of this second bolt is different from the first. Indeed, since there is little left to burn, there is no effect of the second bolt.

In contrast, Fig. 1.13C illustrates the situation where the forest is wet and unburned when the first bolt occurs. It does not cause a major fire, but it does dry out the vegetation so the resulting state is (dry, unburned). Now the second bolt produces a major fire, just as the first bolt did in Fig. 1.13B – since both the state and subsequent input trajectory are the same, the response of the system is the same.

Exercise. A watershed is a region like a valley or basin in which water collects and flows downward toward a river or sea. When it rains heavily the rain water starts to show up quite quickly at a measuring point in the river. For a lighter rain event very little of the rain water may be measured because it is absorbed in the ground. However, after several rain events the ground can get saturated and a light rain can send water quickly downstream. Sketch a set of input/output pairs similar to that of the forest fire to capture the dynamics of such rain events. What variable would you chose to represent that state of the system at the next level of specification.

Exercise. In climates where it can rain or snow, a watershed can have an even more interesting behavior. During winter snow from snow events might accumulate on the ground and there is no indication of any increase in water at the downstream measuring point. But as the temperature increases in spring, the melting snow can eventually cause flooding downstream. Expand your model of the last exercise to include snow events and temperature changes as inputs and sketch the kinds of input/output behaviors you would expect.

1.4.4 COUPLED COMPONENT SYSTEM SPECIFICATION

At the highest level of system specification, we can describe more about the internals of the system. Until now, it was a black box, at first observable only through I/O ports. Subsequently, we were able to peer inside to the extent of observing its state. Now, at level 4, we can specify how the system is composed of interacting components. For example, Fig. 1.14 illustrates how a forest system could be composed of interacting cells, each representing a spatial region, with adjacent cells interconnected. The cells are modeled at level 3, i.e., their state transition and output generation definitions are used to act upon inputs, and generate outputs, respectively to and from, other cells. The cells are coupled together using ports. The output ports of one cell are coupled to the input ports of its neighbors.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 14

B978-0-12-813370-5.00009-2, 00001

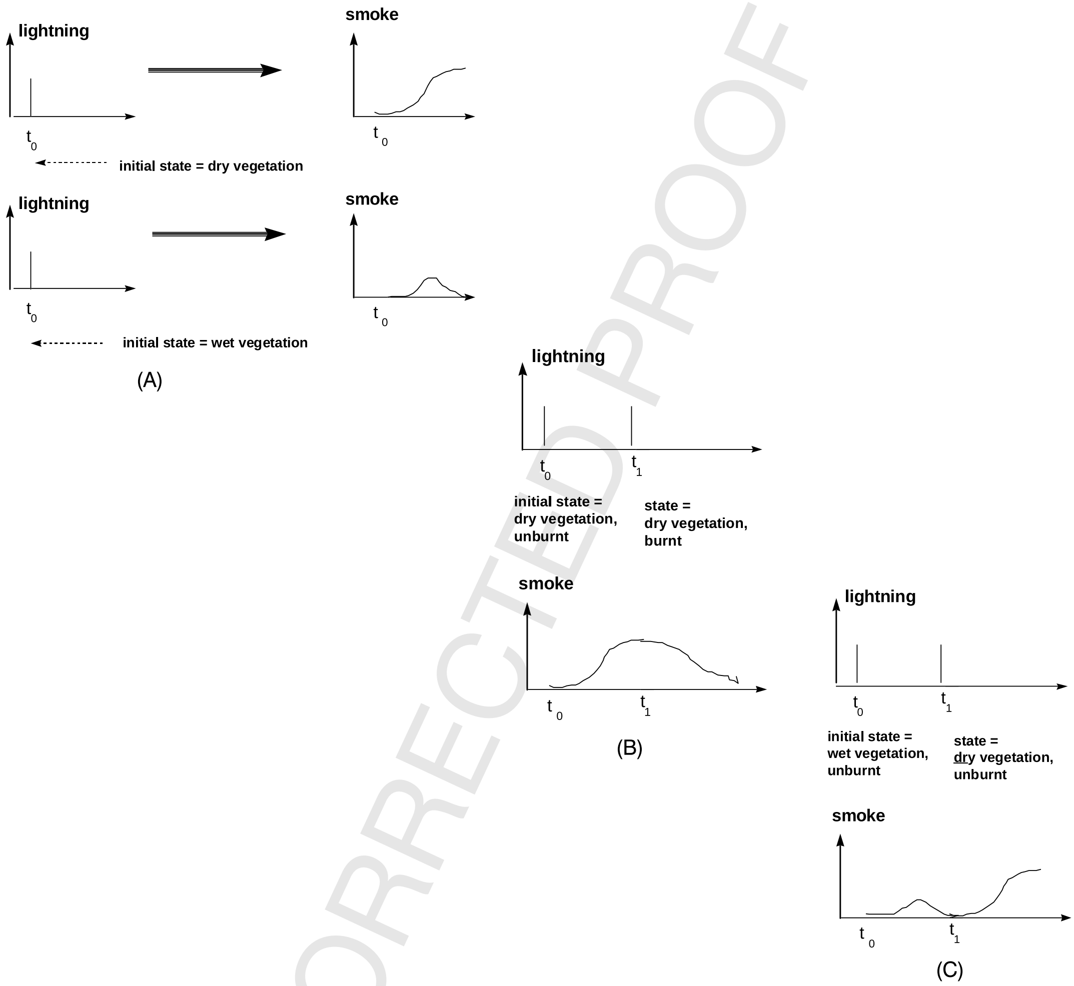

FIGURE 1.13

Initial State Concept: a specification at Level 3 (I/O Function) in which we have initial state knowledge about the forest.

1.5 SYSTEM SPECIFICATION MORPHISMS: BASIC CONCEPTS

The system specification hierarchy provides a stratification for constructing models. But, while constructing models is the basic activity in M&S, much of the real work involves establishing relationships between system descriptions. The system specification hierarchy also provides an orderly way of presenting and working with such relationships. Fig. 1.15 illustrates the idea that pairs of system can be related by morphism relations at each level of the hierarchy. A morphism is a relation that places elements of system descriptions into correspondence as outlined in Table 1.4.

ZEIGLER, 978-0-12-813370-5

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 15

B978-0-12-813370-5.00009-2, 00001

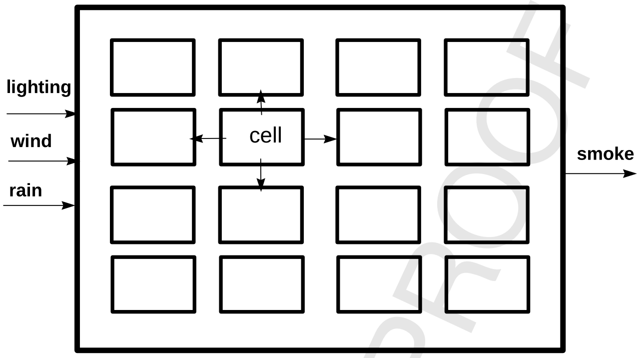

FIGURE 1.14

Component Structure System Specification for the Forrest System.

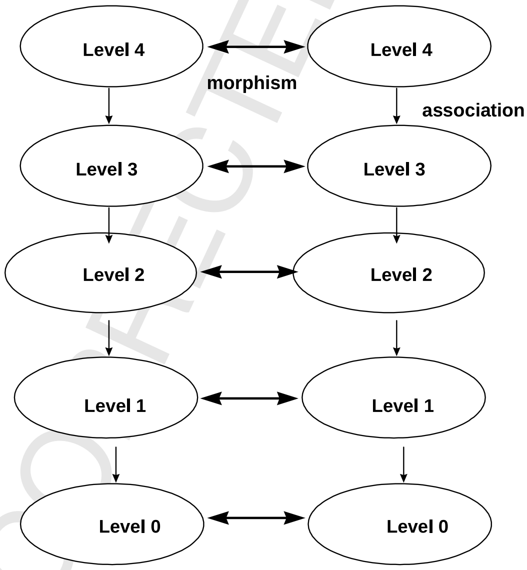

FIGURE 1.15

Morphism Concepts for System Specification Hierarchy.

For example, at the lowest level, two Observation Frames are *morphic*, if we can place their defining elements – inputs, outputs, and time bases into correspondence. Such Frames are *isomorphic* if their inputs, outputs, and time bases respectively, are identical. In general, the concept of morphism tries to capture similarity between pairs of systems at the same level of specification. Such similarity concepts have to be consistent between levels. When we associate lower level specifications with their respective

LATEX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 16

B978-0-12-813370-5.00009-2, 00001

Table 1.4 Morphism relations between systems in System Specification Hierarchy and Klir's levels

Level	Specification Name	Two Systems are Morphic at this level if:
0	Observation Frame	their inputs, outputs and time bases can be put into correspondence
1	I/O Behavior	they are morphic at level 0 and the time-indexed input/output pairs constituting their I/O behaviors also match up in one-one fashion
2	I/O Function	they are morphic at level 0 and their initial states can be placed into correspondence so that the I/0 functions associated with corresponding states are the same
3	State Transition	the systems are homomorphic (explained below)
4	Coupled Component	components of the systems can be placed into correspondence so that corresponding components are morphic; in addition, the couplings among corresponding components are equal

upper level ones, a morphism holding at the upper level must imply the existence of one at the lower level. The morphisms defined in Table 1.4 are set up to satisfy these constraints.

The most important morphism, called **homomorphism**, resides at the State Transition level and is illustrated in Fig. 1.16. Consider two systems specified at level 3, S and S', where S may be bigger than S' in the sense of having more states. Later, we'll see that S could represent a complex model and S' a simplification of it. Or S could represent a simulator and S' a model it is executing. When S' goes

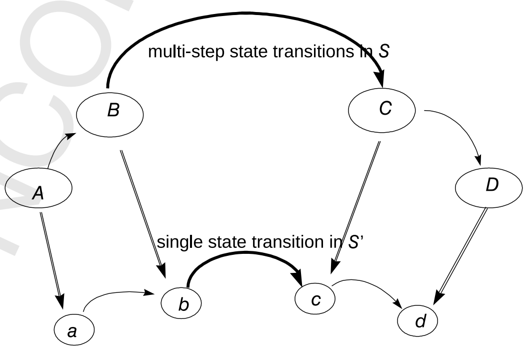

FIGURE 1.16

Homomorphism Concept. This figure illustrates the preservation of state transitions that a homomorphism requires.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 17

B978-0-12-813370-5.00009-2, 00001

through a state sequence such as a, b, c, d, then S should go through a corresponding state sequence say A, B, C, D. Typically, a simulator has a lot of apparatus, represented in its states, necessary to accommodate the whole class of models rather than a single one. Thus we don't assume that states of S and S' are identical – only that there is a predefined correspondence between them illustrated by the shaded connecting lines in the figure. Now to establish that this correspondence is a homomorphism requires that whenever S' specifies a transition, such as from state b to state c, then S actually makes the transition involving corresponding states B and C. Typically, the simulator is designed to take a number of *microstate* transitions to make the *macrostate* transition from B to C. These are computation steps necessary to achieve the desired end result. It is not hard to see that if such a homomorphism holds for all states of S', then any state trajectory in the S' will be properly reproduced in S.

Often, we require that the correspondence hold in a step-by-step fashion. In other words, that the transition from a to b is mirrored by a one-step transition from A to B. Also, as just indicated, we want the I/O Behavior's of homomorphic models specified at the I/O System level to be the same. Thus, as in Fig. 1.17, we require that the outputs produced from corresponding states be the same. In this type of homomorphism, the values and timing of the transitions and outputs of the base model are preserved in the lumped model. Thus, in this case, the state and output trajectories of the two models, when started in corresponding states, are the same.

1.6 EVOLUTION OF DEVS

Around 1960, the first use of a form of digital simulation appeared which we can roughly identify as event-oriented simulation. At its advent, event-oriented simulation was mainly thought to be a form of programming associated with the recent introduction of the digital computer and applied to operational research problems. In contrast, classical simulation was taken to be a form of numerical solution applicable to physics and related sciences whose speed could be greatly increased with mechanical, as opposed to, hand calculation. The concept of "system" was defined by Wymore (1967) as a ba-

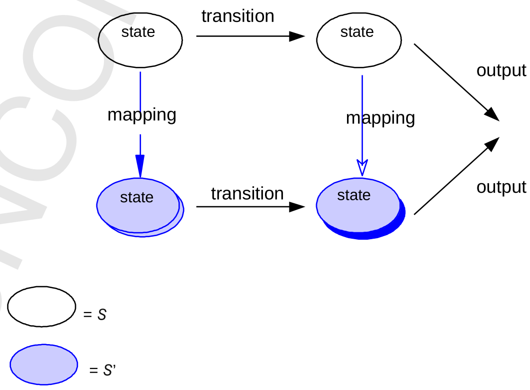

FIGURE 1.17

Homomorphism: a mapping preserving step-by-step state transition and output.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (Irinkeviciute) Prn: 2018/06/22;12:57:08 p. 18

B978-0-12-813370-5.00009-2, 00001

sis for unifying various forms of discrete and continuous model specification. About a decade after event-oriented simulation took hold, the Discrete Event System Specification (DEVS) formalism was defined as a specification for a subclass of Wymore systems that captured all the relevant features of the models underlying event-oriented simulations (Section 1.1.2). In contrast, Discrete Time Systems Specification (DTSS) and Differential Equation System Specification (DESS) were introduced to specify other common distinct subclasses of Wymore systems – the first, as a basis for discrete time models (including those specified by finite automata and cellular automata); the second to represent the continuous models underlying classical numerical solvers. K.D. Tocher appears to be the first to conceive discrete events as the right abstraction to characterize the models underlying the event-oriented simulation techniques that he and others were adopting in the mid-1950s. According to Hollocks (2008), Tocher's core idea conceived of a manufacturing system as consisting of individual components, or 'machines', progressing as time unfolds through 'states' that change only at discrete 'events'. Indeed, DEVS took this idea one step further in following Wymore's formalistic approach, both being based on the set theory of logicians and mathematicians (Whitehead and Russell, 1910, Bourbaki).

Some distinctive modeling strategies soon emerged for programming event-oriented simulation. They became encapsulated in the concept of world views: event scheduling, activity scanning, and process interaction. These world views were formally characterized in Zeigler (1984) showing that they could all be represented as subclasses of DEVS (Chapter 7), thus also suggesting its universality for discrete event model formalisms extending to other representations such as Timed Automata and Petri Nets (Fig. 1.18). Also at the same time the distinction between modular and non-modular DEVS was made showing that the world views all fit within the non-modular category. Moreover, while the modular class was shown to be behaviorally equivalent to that of the non-modular one, it better supported the concepts of modularity, object orientation, and distributed processing that were impending on the software engineering horizon.

An overview of some of the milestones in the development DEVS depicted in the figure below is given in Zeigler and Muzy (2017).

Classic DEVS is a formalism for modeling and analysis of discrete event systems can be seen as an extension of the Moore machine formalism, which is a finite state automaton where the outputs are determined by the current state alone (and do not depend directly on the input). Significantly, the extension associates a lifespan with each state and provides a hierarchical concept with an operation, called coupling, based on Wymore's system theory.

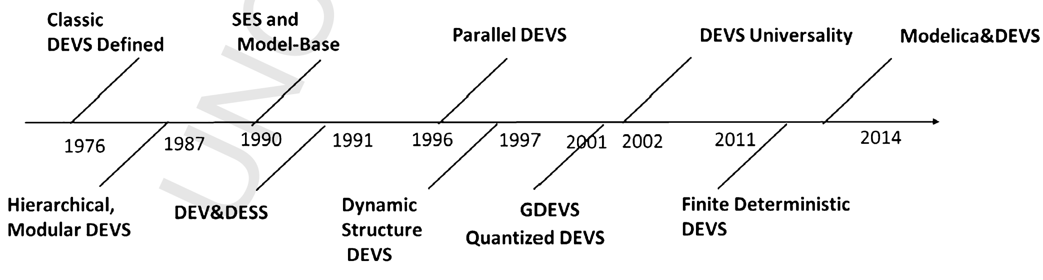

FIGURE 1.18

Timeline of some developments in DEVS.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 19

B978-0-12-813370-5.00009-2, 00001

Parallel DEVS (Chapter 4) revises the classic DEVS formalism to distinguish between transition collisions and ordinary external events in the external transition function of DEVS models, extends the modeling capability of the collisions. The revision also replaces tie-breaking of simultaneously scheduled events by a well-defined and consistent formal construct that allows all transitions to be simultaneously activated providing both conceptual and parallel execution benefits.

Hierarchical, Modular DEVS (Chapter 4) established the similarity and differences with, and implemented DEVS in, the Object-oriented programming (OOP) and modular programming paradigm, among the first in numerous implementations (Van Tendeloo and Vangheluwe, 2017).

System entity structure (SES) (Chapter 18 of TMS2000) is a structural knowledge representation scheme that contains knowledge of decomposition, taxonomy, and coupling of a system supporting model base management.

Dynamic Structure DEVS (Chapter 12), enables representing systems that are able to undergo structural change. Change in structure is defined in general terms, and includes the addition and deletion of systems and the modification of the relations among components.

DEVS considered as a universal computational formalism for systems (Chapter 18) found increasing implementation platforms that handled combined discrete and continuous models (also called co simulation, hybrid simulation, Chapter 12). Some of the milestones in this thread of development are:

- DEV&DESS (Discrete Event and Differential Equation System Specification) (Chapter 9) is a formalism for combined discrete-continuous modeling which based on system theoretical combines the three system specification formalisms-differential equation, discrete time, and the discrete event system specification formalism.
- Quantized State Systems (Chapter 19) are continuous time systems where the variable trajectories are piecewise constant and can be exactly represented and simulated by DEVS. The benefits of this approach in the simulation of continuous time systems are discussed in Chapter 19, including comparisons with conventional numerical integration algorithms in different domain applications.
- GDEVS (Giambiasi et al., 2001) (Generalized DEVS) organizes trajectories through piecewise polynomial segments utilizing arbitrary polynomial functions to achieve higher accuracies in modeling continuous processes as discrete event abstractions.
- Modelica&DEVS (Floros et al., 2011; Nutaro, 2014) transforms Modelica continuous models into DEVS thus supporting models with state and time events that comprise differential-algebraic systems with high index.

A formalism transformation graph showing how the diverse formalism of interest in modeling and simulation can be transformed into DEVS was developed by Vangheluwe (2008).

Finite Deterministic DEVS (Chapter 13) is a powerful subclass of DEVS developed to teach the basics of DEVS that has become the basis for implementations for symbolic and graphical platforms for full-capability DEVS.

This selection of milestones illustrates that much progress has been made. We note the influence of meta-modeling frameworks borrowed from software engineering (OMG, 2015) and increasing applied to development of higher level domain specific languages (Jafer et al., 2016). The confluence of such frameworks with the system-theory based unified DEVS development process (Mittal and Martín, 2013) may be increasingly important in the future simulation model development.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 20

B978-0-12-813370-5.00009-2, 00001

Over the years, DEVS has finding an increasing acceptance in the model-based simulation research community becoming one of the preferred paradigms to conduct modeling and simulation inquiries (Wainer and Mosterman, 2016).

Following the approach proposed in the M&S framework, new variants, extensions and abstractions have been developed using the core of concepts defined by the original formalism.

Several authors have improved the formalism capabilities in response to different situations, giving useful solutions to a wide range of simulation problems.

Some of these solutions (not including listing those in milestones) are Cell-DEVS (Wainer, 2004), Fuzzy-DEVS (Kwon et al., 1996), Min-Max-DEVS (Hamri et al., 2006), and Vectorial DEVS (Bergero and Kofman, 2014).

Moreover, the model/simulator separation of concerns inherent in the M&S framework of Chapter 2 allows researchers to develop alternative simulation algorithms in order to complement existent abstract DEVS simulators (Kim et al., 1998; Muzy and Nutaro, 2005; Shang and Wainer, 2006; Liu and Wainer, 2010).

1.7 SUMMARY

We have outlined the basic concepts of systems theory: structure, behavior, levels of system specification and their associated morphisms. We have brought out the important distinctions that justify having different levels of specification. However, we have not considered all the possible distinctions and levels. For example, the important distinction between modular and non-modular systems has not been recognized with distinct levels. A more complete hierarchy will emerge as revisit the concepts introduced here in a more formal and rigorous manner in Chapter 5. We also have introduced the basic system specification formalisms and outlined the advances in the development of such formalisms that characterize the second edition, TMS2000 and reviewed some of the major milestones in the development of DEVS.

We now turn to a framework for modeling and simulation that identifies the key elements and their relationships. The systems specification hierarchy will provide the basis for presenting this framework. For example, we use specifications at different levels to characterize the different elements. The various system specification formalisms and their simulators provide the operational means to employ the framework in real world applications. We focus on real world application in the last part of the book.

1.8 SOURCES

The basic concepts of systems theory were developed by pioneers such as Arbib (1967), Zadeh and Desoer (1979) (later known more for his fuzzy sets theories), Klir (1985), Mesarovic and Takahara (1975) and Wymore (1977). Since the first edition of this book (Zeigler, 1976) there have been several trends toward deepening the theory (Mesarovic and Takahara, 1989; Wymore, 1993), extending its range of applicability with computerized tools (Pichler and Schwartzel, 1992) and going on to new more abstract formulations (Takahashi and Takahara, 1995). Also, somewhat independently, a new recognition of systems concepts within discrete event systems was fostered by Ho (1992). The DEV&DESS formalism was introduced by Praehofer in his doctoral dissertation (Praehofer, 1991).

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 21

The DEVS Bus originated in the research group of Tag Gon Kim (Kim and Kim, 1996). Quantized system theory was first presented in Zeigler and Lee (1998). A recent collection of systems concepts in computer science is given in Albrecht (1998).

DEFINITIONS, ACRONYMS, ABBREVIATIONS

- DEDS – Discrete Event Dynamic Systems.
- DESS – Differential Equation System Specification.
- DEVS – Discrete Event System Specification.
- DTSS – Discrete Time System Specification.
- DEV&DESS – Discrete Event and Differential Equation System Specification.
- M&S – Modeling and Simulation.
- TMS76 – 1976 Edition of Theory of Modeling and Simulation.
- TMS2000 – 2000 Edition of Theory of Modeling and Simulation.
- TMS2018 – 2018 Edition of Theory of Modeling and Simulation.

REFERENCES

Albrecht, R.F., 1998. On mathematical systems theory. Systems: Theory and Practice, 33–86.

Arbib, M., 1967. Theories of Abstract Automata. Prentice-Hall.

Bergero, F., Kofman, E., 2014. A vectorial DEVS extension for large scale system modeling and parallel simulation. Simulation: Transactions of the Society for Modeling and Simulation International 90 (5), 522–546.

Blas, S.J., Zeigler, B.P., 2018. A conceptual framework to classify the extensions of DEVS formalism as variants and subclasses. In: Winter Simulation Conference.

Floros, X., Bergero, F., Cellier, F.E., Kofman, E., 2011. Automated simulation of Modelica models with QSS methods: the discontinuous case. In: Proceedings of the 8th International Modelica Conference, number 063. March 20th–22nd, Technical University, Dresden, Germany. Linköping University Electronic Press, pp. 657–667.

Giambiasi, N., Escude, B., Ghosh, S., 2001. GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems. In: Proceedings of the 5th International Symposium on Autonomous Decentralized Systems. 2001. IEEE, pp. 464–469.

Hamri, M.E.-A., Giambiasi, N., Frydman, C., 2006. Min–Max-DEVS modeling and simulation. Simulation Modelling Practice and Theory 14 (7), 909–929.

Ho, Y.-C., 1992. Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World. IEEE Press.

Hollocks, B.W., 2008. Intelligence, innovation and integrity—KD Tocher and the dawn of simulation. Journal of Simulation 2 (3), 128–137.

Jafer, S., Chhaya, B., Durak, U., Gerlach, T., 2016. Formal scenario definition language for aviation: aircraft landing case study. In: AIAA Modeling and Simulation Technologies Conference, p. 3521.

Kim, K.H., Kim, T.G., Park, K.H., 1998. Hierarchical partitioning algorithm for optimistic distributed simulation of DEVS models. Journal of Systems Architecture 44 (6–7), 433–455.

Kim, Y.J., Kim, T.G., 1996. A heterogeneous distributed simulation framework based on DEVS formalism. In: Proceedings of the Sixth Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems, pp. 116–121.

Klir, G.J., 1985. Architecture of Systems Complexity. Saunders, New York.

Kwon, Y., Park, H., Jung, S., Kim, T., 1996. Fuzzy-DEVS formalism: concepts, realization and applications. In: Proceedings AIS 1996, pp. 227–234.

Liu, Q., Wainer, G., 2010. Accelerating large-scale DEVS-based simulation on the cell processor. In: Proceedings of the 2010 Spring Simulation Multiconference. Society for Computer Simulation International, p. 124.

Mesarovic, M.D., Takahara, Y., 1975. General Systems Theory: Mathematical Foundations, vol. 113. Academic Press.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 22

B978-0-12-813370-5.00009-2, 00001

Mesarovic, M., Takahara, Y., 1989. Abstract Systems Theory, vol. 116. Springer-Verlag, NY.
Mittal, S., Martín, J.L.R., 2013. Netcentric System of Systems Engineering with DEVS Unified Process. CRC Press.
Muzy, A., Nutaro, J.J., 2005. Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators. In: 1st Open International Conference on Modeling & Simulation. OICMS, pp. 273–279.
Nutaro, J., 2014. An extension of the OpenModelica compiler for using Modelica models in a discrete event simulation. Simulation 90 (12), 1328–1345.
OMG, 2015. Documents associated with meta object facility version 2.5. Available via http://www.omg.org/spec/MOF/2.5/. (Accessed 2 November 2016).
Pichler, F., Schwartzel, H., 1992. CAST (Computer Aided System Theory) Methods in Modeling. Springer-Verlag, New York.
Praehofer, H., 1991. System theoretic formalisms for combined discrete-continuous system simulation. International Journal of General System 19 (3), 226–240.
Shang, H., Wainer, G., 2006. A simulation algorithm for dynamic structure DEVS modeling. In: Proceedings of the 38th Conference on Winter Simulation. Winter Simulation Conference, pp. 815–822.
Takahashi, S., Takahara, Y., 1995. Logical Approach to Systems Theory. Springer-Verlag, London.
Van Tendeloo, Y., Vangheluwe, H., 2017. An evaluation of DEVS simulation tools. Simulation 93 (2), 103–121.
Vangheluwe, H., 2008. Foundations of modelling and simulation of complex systems. Electronic Communications of the EASST 10.
Wainer, G.A., 2004. Modeling and simulation of complex systems with cell-DEVS. In: Proceedings of the 36th Conference on Winter Simulation. Winter Simulation Conference, pp. 49–60.
Wainer, G.A., Mosterman, P.J., 2016. Discrete-Event Modeling and Simulation: Theory and Applications. CRC Press.
Whitehead, A., Russell, B., 1910. Principia Mathematica 1, 1 ed. Cambridge University Press, Cambridge. JFM 41.0083.02.
Wymore, A.W., 1993. Model-Based Systems Engineering, vol. 3. CRC Press.
Wymore, W., 1967. A Mathematical Theory of Systems Engineering: The Elements. Wiley.
Wymore, W., 1977. A Mathematical Theory of Systems Engineering: The Elements. Krieger Pub Co.
Zadeh, L.A., Desoer, C.A., 1979. Linear System Theory. Krieger Publishing Co.
Zeigler, B.P., 1976. Theory of Modeling and Simulation. Wiley Interscience Co.
Zeigler, B., Muzy, A., 2017. From discrete event simulation to discrete event specified systems (DEVS). IFAC-PapersOnLine 50 (1), 3039–3044.
Zeigler, B.P., 1984. Multifacetted Modelling and Discrete Event Simulation. Academic Press Professional, Inc, London.
Zeigler, B.P., Lee, J.S., 1998. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. In: SPIE Proceedings, vol. 3369, pp. 49–58.

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 23

NON-PRINT ITEMS

Abstract

We outline basic concepts of systems theory: structure, behavior, levels of system specification and their associated morphisms. We bring out the important distinctions that justify having different levels of specification. A more complete hierarchy will emerge as we revisit the concepts introduced in a more formal and rigorous manner later. We also introduce the basic system specification formalisms and outline the recent advances in the development of such formalisms.

Keywords

System specification, System specification morphism, Levels of system specification, Systems modeling, DEVS, Discrete event system specification, Differential equation system specification

LaTeX class: zeigler, 2018/05/10 08:43:53 zeigler driver
ZEIGLER[monography] model: p05 • color: 1c • ref-style: Harvard Name-Year system. (Style 2) • lang: en-US • docsubty: CHP •
file: zeigler01.tex (lrinkeviciute) Prn: 2018/06/22;12:57:08 p. 24