Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

INTRODUCTION TO ITERATIVE SYSTEM COMPUTATIONAL FOUNDATIONS AND DEVS

Bernard P. Zeigler

RTSync Corp.
6909 W Ray Rd STE 15-107
Chandler, AZ 85226-1526, USA

ABSTRACT

Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations (Third Edition) follows its predecessors in providing an authoritative and complete theoretical solution for researchers, graduate students and working engineers interested in posing and solving problems using the tools of computational modeling and simulation. We discuss its main unifying concept, the iterative systems specification, that enables models at any level of structure or behavior, to be transparently mapped into the Discrete Event System Specification (DEVS) formalism. Emphasizing the integration of discrete event and continuous modeling approaches, iterative systems specification provides a sound system-theoretic basis for quantized system and distributed co-simulation supporting the co-existence of multiple formalisms in model components. This introductory exposition emphasizes basic intuitive concepts rather than theorems and proofs that support them. As a running example, we apply the iterative systems formalism to bridging the cognitive behavior to neural circuits gap.

1 MODELING AND SIMULATION FRAMEWORK (MSF)

Review of Iterative System Computational Foundations and DEVS begins with review of systems concepts and the basic entities and relations of Modeling and Simulation in Chapters 1 and 2 of Theory of Modeling and Simulation (Zeigler et al. 2018), hereafter TMS 2018.

1.1 System concepts

The US Department of Defense M&S Glossary (M&SCO, 2016) gives these definition: (1) *system model*: A representation of a system; and (2) *simuland*: The system being simulated by a simulation. These define "system model" and "simuland" in terms of "system" but nowhere to be found is a definition of "system" itself. In contrast, the Modeling and Simulation Framework (MSF), to be reviewed here, includes the "system" as one of the four basic entities along with "experimental frame", "model" and "simulator". Moreover, we need some basic concepts about systems that are reviewed only in outline.

1.2 System Specification Hierarchy Levels of System Specification

The MSF sets forth the fundamental entities and relationships in the M&S enterprise. To describe these items we employ the systems specification hierarchy as the basis for the MSF. Table 1 identifies five basic levels of system specification forming a Systems Specification Hierarchy. The fourth column gives an example of a system specification at each level applied to a person in a conversation. Later we will formulate a conversation, itself, as system composed of two interacting persons. At each level we know some important things about a system that we didn't know at lower levels. At the lowest level, 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).

The System Specification Hierarchy 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. 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.

Table 1: Levels of system specification.

Level	**Specification Name**	**What we know at this level**	**Example: A Person in a Conversation**
0	Observation Frame	How to stimulate the system with inputs; what variables to measure and how to observe them over a time base;	The person has inputs and outputs at the usual cognitive level, such as streams of words
1	I/O Behavior	Time-indexed data collected from a source system; consists of input/output pairs	For each input that the person recognizes, the set of possible outputs that the person can produce
2	I/O Function	Knowledge of initial state; given an initial state, every input stimulus produces a unique output.	Assuming knowledge of the person's initial state when starting the conversation, the unique output response to each input.
3	State Transition	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.	How the person transits from state to state under input words and generates output words from the current state
4	Coupled Component	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.	A description of a person's I/O behavior in terms of neural components and their interaction by spikes is at this level.

1.3 The Entities and Relations of the Modeling and Simulation Framework

As illustrated in Figure 1, the basic entities of the framework are: *source system, model*, *simulator* and *experimental frame*. The basic inter-relationships among entities are the modeling and the simulation relationships. The entities are defined in Table 2 which also characterizes the level of system specification that typically describes the entities. The level of specification is an important feature for distinguishing

between the entities, which is often confounded in the literature. You can return to Figure 1 and Table 2 to keep an overall view of the framework as we describe each of the components in the following presentation.

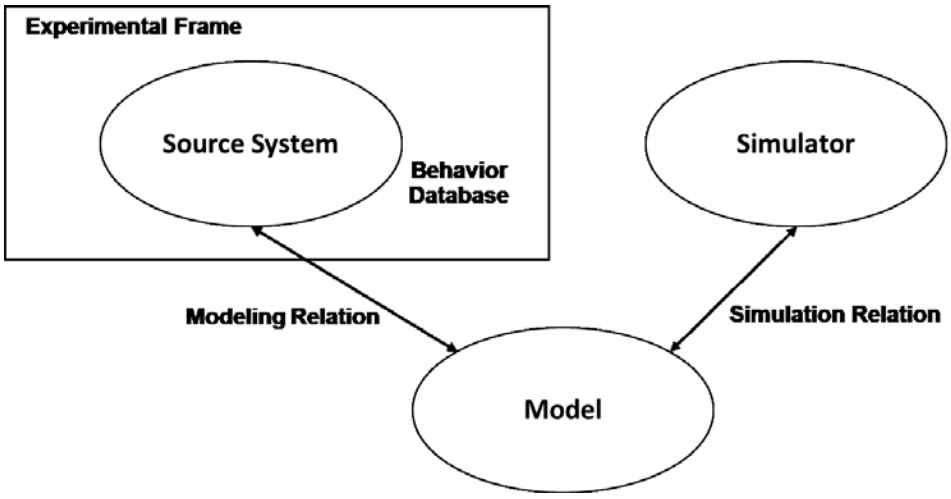

Figure 1: Fundamental entities and relationships in the M&S framework.

1.3.1 Source System

The *source system* (we will omit the 'source' qualifier, when the context is clear) is the real or virtual environment that we are interested in modeling. It is viewed as a *source of observable data*, in the form of time-indexed trajectories of variables. The data that has been gathered from observing or otherwise experimenting with a system is called the *system behavior database*. As indicated in 0, this concept of system is a specification at level 0 and its database is a specification at level 1. This data is viewed or acquired through experimental frames of interest to the modeler.

Table 2: Defining the basic entities in M&S and their usual levels of specification.

Basic Entity	**Definition**	**Related System Specification Levels**
Source system	Real or artificial source of data	Known at level 0
Behavior database	Collection of gathered data	Observed at level 1
Experimental frame	Specifies the conditions under which system is observed or experimented with	Constructed at levels 3 and 4
Model	Instructions for generating data	Constructed at levels 3 and 4
Simulator	Computational device for generating behavior of the model	Constructed at level 4

1.3.2 Experimental Frame

An *experimental frame* is a specification of the conditions under which the system is observed or experimented with. As such an experimental frame is the operational formulation of the objectives that motivate a modeling and simulation project. For example, out of the multitude of variables that relate to a forest, the set {lightning, rain, wind, smoke} represents one particular choice. Such an experimental frame

is motivated by the interest in modeling the way lightning ignites a forest fire. A more refined experimental frame would add the moisture content of the vegetation and the amount of unburned material as variables. Thus, many experimental frames can be formulated for the same system (both source system and model) and the same experimental frame may apply to many systems. Why would we want to define many frames for the same system? Or apply the same frame to many systems? For the same reason that we might have different objectives in modeling the same system, or have the same objective in modeling different systems. More of this in a moment.

There are two equally valid views of an experimental frame. One, views a frame as a definition of the type of data elements that will go into the database. The second views a frame as a system that interacts with the system of interest to obtain the data of interest under specified conditions. In this view, the frame is characterized by its implementation as a measurement system or observer. In this implementation, a frame typically has three types of components (as shown in Figure 2): *generator that* generates input segments to the system; *acceptor* that monitors an experiment to see the desired experimental conditions are met; and *transducer* that observes and analyzes the system output segments.

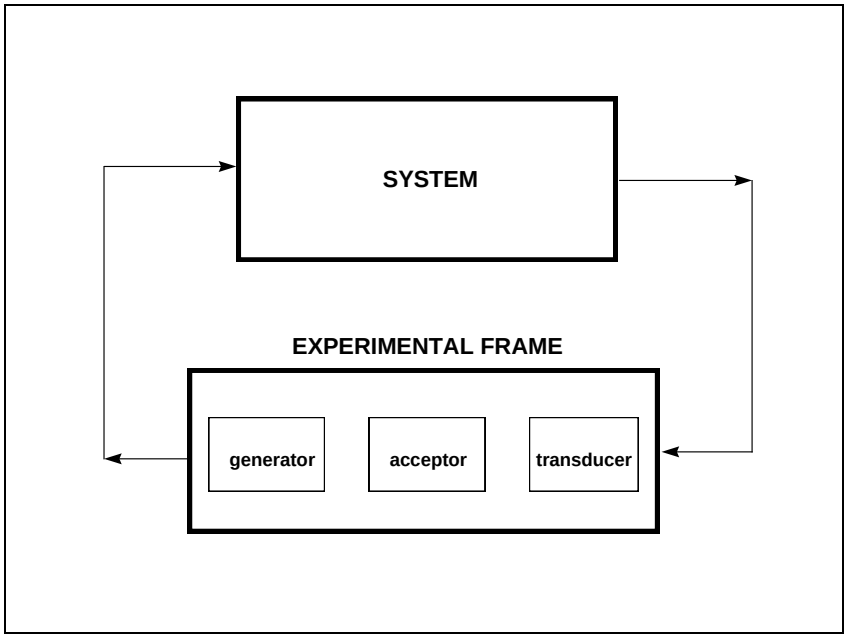

Figure 2: Experimental frame and its components.

1.3.3 Objectives and Experimental Frames

Objectives for modeling relate to the role of the model in systems design, management or control. The statement of objectives serves to focus model construction on particular issues. See TMS2018 Chapter 2 (Zeigler et al. 2018) for discussion of a process for transforming objectives into experimental frames.

1.3.4 Example: Two Person Interaction

The conversation example for the System Specification Hierarchy of Table 1 actually assumes an underlying experimental frame that was not completely specified. We can specify the objective of trying to characterize when a two person interaction is a valid conversation, i.e., when the participants exchange words that make sense to an observer. We restrict the interaction to an exchange of greetings of two people passing by each other. There are relatively few pairs of words that make sense such as "Hello, Hi" as a

greeting by one person and a response of the other, whereas most other pairs of words do not. Such pairs are in effect a description at the I/O Behavior level for the interaction of two persons. An experimental frame in this case centers on collecting such I/O pairs in both a real two-person encounter and a simulation model of it. The acceptor component of such a frame can monitor the interaction for pairs judged to be indicative of a valid conversation.

1.3.5 Model

In its most general guise, a *model* is a system specification at any of the levels of the System Specification Hierarchy. However, in the traditional context of M&S, the system specification is usually done at levels 3 and 4. Thus the most common concept of a simulation model is that it is a *set of instructions, rules, equations, or constraints for generating I/O behavior*. In other words, we write a model with a state transition and output generation mechanisms (level 3) to accept input trajectories and generate output trajectories depending on its initial state setting. Such models form the basic components in more complex models that are constructed by coupling them together to form a level 4 specification.

1.3.6 Example (Continued): Two Person Interaction

An example of a conversation between two persons can be modeled as agents interacting through exchange of messages carried by discrete events. This constitutes a coupled model with atomic model components. Each component alternates between speaking and listening phases. If moreover, the components adhere to the discipline that only one is in the speaking phase at any time, then the result represents a valid conversation. If at any time, both components are in the same phase (speaking or listening) then the experimental frame acceptor just discussed will stop the simulation and declare this as an invalid conversation.

There are many meanings that are ascribed to the word "model". For example, a model is conceived as any physical, mathematical, or logical representation of a system, entity, phenomenon, or process. *The definition in terms of system specifications has the advantages that it has a sound mathematical foundation and is has a definite semantics that everyone can understand in unambiguous fashion.* Like other formal definitions, it cannot capture all meanings in the dictionary. However, it is intended to capture the most useful concepts in the M&S context.

1.3.7 Simulator

As a set of instructions, a model needs some agent capable of actually obeying the instructions and generating behavior. We call such an agent a *simulator*. Thus, a simulator is any computation system (such as a single processor, a processor network, the human mind, or more abstractly an algorithm), capable of executing a model to generate its behavior. A simulator is typically specified at a high level since it is a system that we design intentionally to be synthesized from components that are off-the-shelf and well-understood. Separating the model and simulator concepts provides a number of benefits for the framework:

- The same model, expressed in a formalism, may be executed by different simulators thus opening the way for portability and interoperability at a high level of abstraction.
- Simulator algorithms for the various formalisms may be formulated and their correctness rigorously established.
- The resources required to correctly simulate a model afford a measure of its complexity.

1.3.8 Example Continued: Two Person Interaction

The two-person discrete event model just mentioned can be formulated within the Discrete Event System Specification (DEVS) formalism as illustrated in Chapter 10 of TMS 2018 (Zeigler et al. 2018). This then

allows the model behavior to be generated by a DEVS simulator, i.e., a simulation program implementing the rules of the Abstract DEVS simulation protocol that guarantees correct simulation of any DEVS model.

Table 3: Primary Relationships among Entities.

Basic Relationship	**Definition**	**Related System Specification Levels**
Modeling relation Replicative validity Predictive validity Structural validity	Concerned with how well model-generated behavior agrees with observed system behavior	Comparison is at level 1 Comparison is at level 2 Comparison is at level 3,4
Simulation relation Correctness	Concerned with assuring that the simulator carries out correctly the model instructions	Basic comparison is at level 2; involves homomorphism at levels 3 or 4

2 PRIMARY RELATIONS AMONG ENTITIES

The entities - *system, experimental frame, model, and simulator* – become truly significant only when properly related to each other. For example, we build a model of a particular system for some objective - only some models, and not others, are suitable. Thus, it is critical to the success of a simulation modeling effort that certain relationships hold. The two most fundamental are the modeling and the simulation relations (0)

2.1 Modeling Relation: Validity

The basic modeling relation, *validity,* refers to the relation between a model, a system and an experimental frame. Validity is often thought of as the degree to which a model faithfully represents its system counterpart. However, it makes much more practical sense to require that the model faithfully captures the system behavior only to the extent demanded by the objectives of the simulation study. In the MSF, the concept of validity answers the question of whether it is impossible to distinguish the model and system in *the experimental frame of interest.* The most basic concept, *replicative validity*, is affirmed if, for all the experiments possible within the experimental frame, the behavior of the model and system agree within acceptable tolerance. Thus replicative validity requires that the model and system agree at the I/O relation level 1 of the system specification hierarchy.

Stronger forms of validity are *predictive validity* and structural *validity*. In predictive validity we require not only replicative validity, but also the ability to predict as yet unseen system behavior. To do this the model needs to be set in a state corresponding to that of the system. Thus predictive validity requires agreement at the next level of the system hierarchy, that of the I/O function level 2. Finally, *structural validity* requires agreement at level 3 (state transition) or higher (coupled component). This means that the model not only is capable of replicating the data observed from the system but also mimics in step-by-step, component-by-component fashion, the way that the system does its transitions.

The term *accuracy* is often used in place of validity. Another term *fidelity,* is often used for a combination of both validity and detail. Thus, a high fidelity model may refer to a model that is both high in detail and in validity (in some understood experimental frame). However when used this way, beware that there may be a tacit assumption that high detail alone is needed for high fidelity, as if validity is a necessary consequence of high detail. In fact, it is possible to have a very detailed model that is nevertheless very much in error, simply because some of the highly resolved components function in a different manner than their real system counterparts.

2.2 Simulation Relation: Simulator Correctness

The basic *simulation* relation, *simulator correctness*, is a relation between a *simulator* and a *model.* A *simulator correctly simulates a model* if it is guaranteed to faithfully generate the model's output trajectory given its initial state and its input trajectory. Thus, simulator correctness requires agreement at the I/O function level 2. In practice, simulators are constructed to execute not just one model but a family of possible models. This flexibility is necessary if the simulator is to be applicable to a range of applications. In such cases, we must establish that a simulator will correctly execute a particular class of models. Since the structures of both the simulator and the model are at hand, it may be possible to prove correctness by showing that a *homomorphism* relation holds. Here a homomorphism is a correspondence between simulator and model states that is preserved under transitions and outputs.

2.3 Modeling as Valid Simplification

The inescapable fact about modeling is that it is severely constrained by complexity limitations. Complexity, is at heart, an intuitive concept – the feeling of frustration or awe that we all sense when things get too numerous, diverse, or intricately related to discern a pattern, to see all at once – in a word, to comprehend. Generalizing from the boggled human mind to the overstressed simulator suggests that the complexity of model can be measured by the resources required by a particular simulator to correctly interpret it. As such, complexity is measured relative to a particular simulator, or class of simulators. However, properties intrinsic to the model are often strongly correlated with complexity independently of the underlying simulator. Successful modeling can then be seen as *valid simplification*. We need to *simplify,* or reduce the complexity, to enable our models to be executed on our resource-limited simulators. But the simplified model must also be *valid*, at some level, and within some experimental frame of interest. Note tat there is always a pair of models involved, call them the *base* and *lumped* models. Here, the base model is typically "more capable" and requires more resources for interpretation than the lumped model. By the term "more capable", we mean that the base model is valid within a larger set of experimental frames (with respect to a real system) than the lumped model. However, the important point is that within a *particular frame of interest* the lumped model might be just as valid as the base model. The concept of morphism affords criteria for judging the equivalence of base and lumped models with respect to an experimental frame.

3 ITERATIVE SYSTEM SPECIFICATIONS

Iterative system specification is intended to be an abstraction that provides an approach to hybrid M&S at the transdisciplinary level. This abstraction is more fundamental than DEVS in that it serves as a precursor that provides representations for the preimages of reality that are eventually presented as crisp discrete events. In fact, DEVS has abstracted away the original continuous and fuzzified boundaries begging the question of how can one come up with the events, states, and components in the next problem to be tackled. Iterative systems specification helps restore more of the left out reality. This provides a more nuanced approach to hybrid M&S at a fundamental systems level that can then be implementable in the computational medium of DEVS. In other words, iterative systems specification offers a constructive approach to general systems development. The approach is to combine Effective Computation Theory with Systems Theory in a way that allows us to specify systems in an event-like computational manner that is ultimately implementable in DEVS-based M&S environments (see TMS 2018 Chapter 10.)

3.1 Input Generators

Input generators are relatively short segments (time functions) that can be concatenated to form longer segments. For example, Figure 3(a) shows three types of generators: 1) null segments that can last any finite duration, 2) constant segments that can last any finite duration and 3) pulse segments lasting at most a duration T (e.g., 10 seconds). As illustrated in Figure 3(b), we consider composite inputs segments

composed of concatenated generators placed one after another end-to-end. Such segments can be parsed into their component parts by a process called maximal length segmentation (MLS). With input segments broken into long duration generators we can generate the state trajectories associated with the system starting in its different states taking jumps over long time intervals as shown in Figure 3(c). To do this we need the transition function to be given in terms of the generators so that given a starting state and an input generator we compute a state at the end of the generator. Doing it this way can save a lot of computation that would otherwise be done with small steps as suggested in the figure. Of course, it might be necessary to do the small step computation to compile the state transition function for the generators. There is still much utility in doing this, since the compiled form can be used every time the generator appears in an input segment. Also if there is an analytic solution for the generator input's effect on the state then we can use that within the large step-based computation. Since the output trajectory is computed from that state trajectory, we can compute the output trajectories corresponding to the input generators as well.

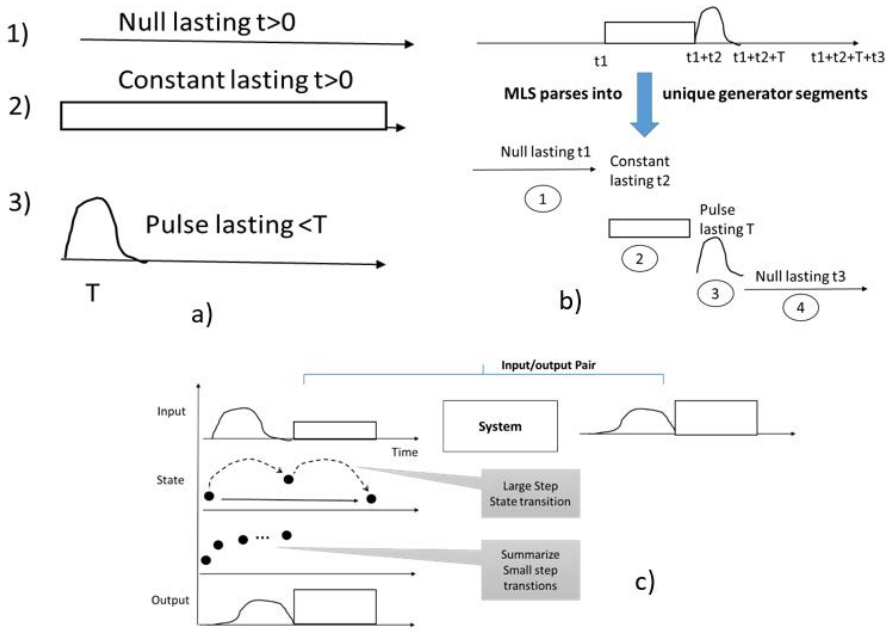

Figure 3: (a) Illustrating input generators, (b) Maximal Length Segmentation, (c) Computing with generators.

The computation for atomic and coupled models can roughly be described as follows:

Atomic component system – compute I/O function

1. Apply MLS to get finite sequence of input generators
2. Given starting state, apply the transition function and output function to get next state and output for successive generators
3. Collect input/output pairs into I/O function

Composition of component systems

1. Compute I/O function of each component
2. Given starting states, guess input generators

3. Apply I/O functions to get output generators
4. Apply the coupling relations, to get input generators
5. Repeat step 3 until consistent input/output relation is obtained
6. Advance time using the minimum MLS break point (where shortest generator ends) at each stage

Note that for these computations to produce effective results they must terminate. We shall now see that the particular conditions that guarantee such effective computation will be called *progressivity* and *well-definition*

3.2 Progressivity and Well-definition of Systems

Closure under coupling means that the resultant of a composition of systems is itself a system. At the coupled level of system specification, the state of the system is a vector-like listing of its component states. So now let's look at this vector-like space abstractly as a set of states and transitions. This leads us to a concept of an input free system (or autonomous, i.e., not responding to inputs) with outputs not considered and with states, transitions, and times associated with transitions. In this view, we have a transition system

$$M = < S, \delta, ta >, \text{ where } \delta \subseteq S \times S, \quad \text{and } ta : S \times S \rightarrow R_0^\infty$$

where, S is set of states, δ is the transition function and *ta* is the time advance. Note, we are allowing the transition system to be nondeterministic so that rather than being a function it is presented as a relation. This recognizes the fact that our original composition of systems may not have deterministic transitions. In Figure 4, there are transitions from state S1 to state S3 and from S3 to S4, each of which take 1 time unit and there is a cycle of transitions involving S4…S7 each of which take zero time. There is a self-transition involving S2 which consumes an infinite amount of time (signifying that it is passive, remaining in that state forever.) This is distinguished from the absence of any transitions out of S8.

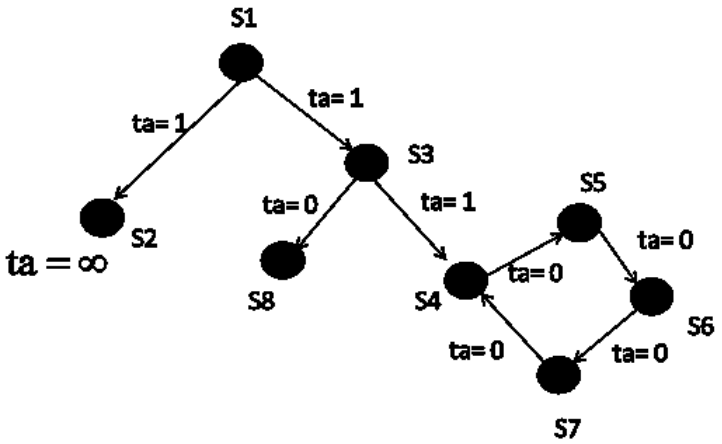

Figure 4. Example non-deterministic timed transition system.

This example gives us a quick understanding of the conditions for system existence at the fundamental level: We say that the system is:

- *Not defined* at a state, e.g., S8, because there is no trajectory emerging from it. More precisely, there is no transition pair with the state as its left member, i.e., there is no pair (S8, x) for x in S.
- *Non-deterministic* at a state, e.g., S1, because there are two distinct outbound transitions defined for it. More precisely, the state is a left member of two transition pairs, e.g., for S1 we have (S1, S2) and (S1,S3).

- *Deterministic* at a state when there is exactly one outbound transition. For example, S2 and S4 are deterministic because there is only one outbound transition for each.

We say that the system is *well-defined* if it is defined at all its states. These conditions are relative to static properties, i.e., they relate to states per-se not how the states follow one another over time. In contrast, state trajectories (i.e., sequences of states following along existing transitions, e.g., S1, S3, S4) relate to dynamic and temporal properties. When moving along a trajectory we keep adding the time advances to get the total traversal time, e.g., the time taken to go from S2 to S4 is 2. Here a trajectory is said to be *progressive* in time if time always advances as we extend the trajectory. For example, the cycle of states S4…S7 is not progressive because as we keep adding the time advances the sum never increases. Conceptually, let's start a clock at 0 and, starting from a given state, we let the system evolve following existing transitions and advancing the clock according to the time advances on the transitions. If we then ask what the state of the system will be at some time later, we will always be able to answer if the system is well-defined and progressive. A well-defined system that is not progressive signifies that the system gets stuck in time and after some time instant, it becomes impossible to ask what the state of the system is in after that instant. Zeno's paradox offers a well-known metaphor where the time advances diminish so that time accumulates to a point rather than continues to progress. It also offers an example showing that the pathology does not necessarily involve a finite cycle. This concept of progressiveness generalizes the concept of legitimacy for DEVS (TMS2018 Chapter 6).

Thus we have laid the conceptual groundwork in which a system has to be *well-defined* (static condition) and *progressive* (temporal dynamic condition) if it is to be a bonafide result of a composition of components.

3.3 Active/passive Compositions

An interesting form of the interaction among components may take on a pattern found in numerous information technology and process control systems. In this interaction, each component system alternates between two phases, active and passive. When one system is active the other is passive only one can be active at any time. The active system does two actions: 1) it sends an input to the passive system that activates it (puts it into the active phase), and 2) it transits to the passive phase to await subsequent reactivation. For example, in a Turing Machine (TMS2018 Chapter 4), the TM control starts a cycle of interaction by sending a symbol and move instruction to the tape system then waiting passively for a new scanned symbol to arrive. The tape system waits passively for the symbol/move pair. When it arrives it executes the instruction and sends the symbol now under the head to the waiting control.

Active/passive compositions provide a class of systems from which we can draw intuition and examples for generalizations about system emergence at the fundamental level. TMS2018 employs the pattern of active/passive compositions to illuminate the conditions that result in ill definition of deterministic, nondeterministic and probabilistic systems. They provide sufficient conditions, meaningful especially for feedback coupled assemblages, under which iterative system specifications can be composed to create a well-defined resultant system.

3.4 How can feedback coupled components define a system?

We examine the problem of emergence of a well-defined system from a coupling of a pair of component systems (the smallest example of the multicomponent case). As in Figure 5, the time base is a critical parameter to take into account on which input and output streams (functions of time) are happening. The cross-coupling imposes a pair of constraints as shown that the output time function of one component must equal the input time function of the other. The problem is given that the system components have fixed input/output behavior how can the constraints be solved so that their composition forms a well-defined system? We must recognize that inputs and outputs are occurring simultaneously in continuous time so that the constraints must be simultaneously satisfied at every instant. One step toward a solution is to break the time functions into parts or segments that allow restricting the problem to segments rather than complete

streams. We can apply the insight from the discussion of well-definition for timed transition systems (Figure 4) to the coupled system case by recognizing that the abstract state of the latter represented the *vector-like* state of the former. This leads us to say that for any pair of states (s1, s2) of components S1 and S2 , determinism requires that there be a unique trajectory emerging from it. This in turn, requires that for all generator segments, 1) the I/O relations are satisfied and 2) the coupling relations are satisfied. Furthermore, the time advances obtained by the maximal length segmentations as we go from one generator to the next must satisfy the progressivity requirement.

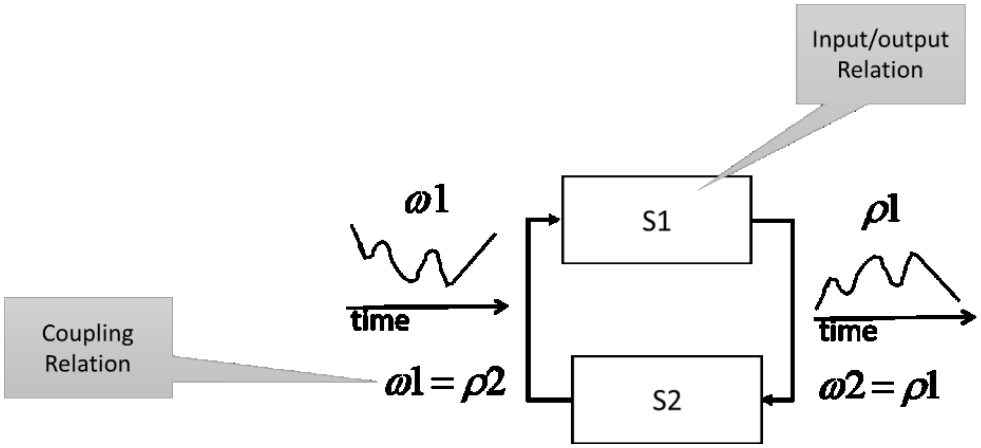

Figure 5: Interaction of two components attempting to form a composite system.

For active/passive compositions, TMS2018 Chapter 10 shows that input/output constraints are satisfied as long as the active component's output segment is accepted by the passive component and that component outputs only null segments while passive. An example where active/passive composition may solve a part of the enigma of how humans came to develop conversational language is discussed in (Mittal and Zeigler 2014).

4 SIMULATION OF ITERATIVE SYSTEM SPECIFICATION BY DEVS

A full proof that any iterative system specification can be simulated by a DEVS is found in TMS2018 Chapter 20. In outline, an exact simulation can be had by encoding an input segment as a DEVS event segment with the event capturing the segment itself and its length equal to that of the segment. The DEVS uses the generator transition function of the iterative specification to compute the next state and generate the corresponding output segment as an event. After holding for a time equal to the segment length, the DEVS receives the next encoded segment and the cycle continues. More parsimonious encoding are possible. For example, the initial and final values of an input segment can be encoded as the event and with its length. This is enough information to compute the *activity* of an overall segment which is the sum of the absolute differences in these pairs (TMS2018 Chapter 10). Depending on the iterative specification, the DEVS simulation might prove to provide an acceptable approximation to the activity of the overall output segment and of the complete I/O relation.

4.1 Class Mapping of Iterative System Specifications

As indicated the focus of TMS2018 is on iterative systems specification and it considers several specializations and elaborations that show it is a) a constructive formalism for general systems specification and b) computationally implementable as extension to DEVS-based modeling and simulation environments. Figure 6 maps out some of these variations and places them in relation to each other and to the general system. Two relations are shown. One in which a formalism *simulates* another and the other in

which one *specifies* another. We have seen that DEVS *simulates* iterative systems specifications in the sense that given an iterative systems specification, there is a DEVS that can simulate it.

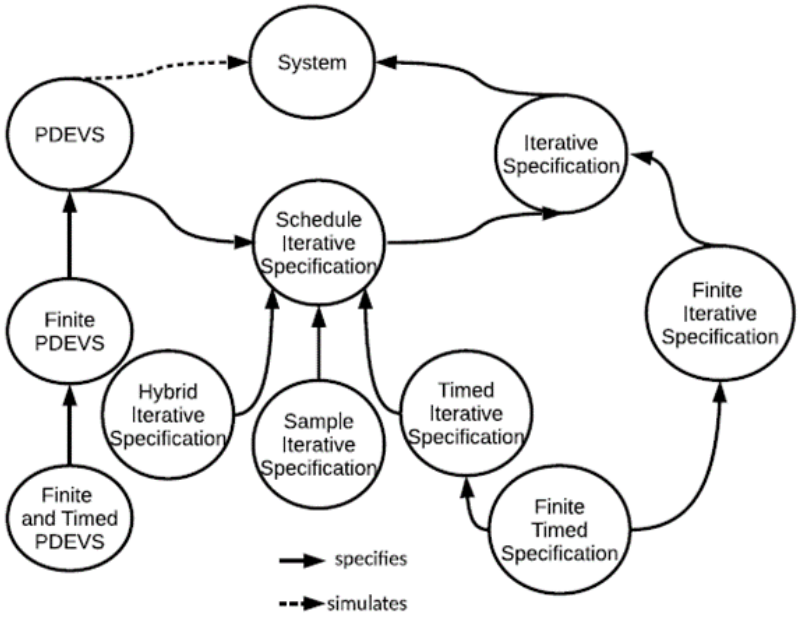

Figure 6. Class mapping of iterative system specifications.

Besides *PDEVS* (Parallel Discrete Event Systems Specification), several other iterative specifications are shown in Figure 6. *Schedule Iterative Specification*, introduces a time scheduling concept explicitly into the basic iterative systems specification. This allows the modeler to employ a time advance function that generalizes the time advance concept of PDEVS. It does not extend the class of systems that are specified since the modeler can already define such a function on a case-by-case basis to determine when an internal event should happen. In contrast, the time advance function in Schedule Iterative Specification provides this capability explicitly giving it a well-defined semantics. PDEVS then turns out to be a specific kind of Schedule Iterative Specification. Two formalisms are then shown as further specializations of Schedule Iterative Specification. Hybrid Iterative Specification is a formulation of the Hybrid system specification to support hybrid co-simulation (TMS 2018 Chapter 9). Similarly, Sample Based Iterative Specification is a formulation of the HyFlow formalism (Barros 2002) which allows a system to sample its input in a single atomic operation, a capability that can be conceptually and operationally appealing. This facility can be captured within the Schedule Iterative Specification , but not directly within PDEVS. Indeed, PDEVS lacks the ability to directly "pull" a sample value from a source; it must instead request and wait in a two-step sequence. However, in keeping with PDEVS' ability to simulate any iterative specification, it can simulate the Sample Based Iterative Specification based implementation of the request/wait equivalence in the abstract simulator. Finite versions of these specifications are also defined,

5 SUPPORT FOR CO-EXISTENCE OF MULTIPLE FORMALISMS AND DISTRIBUTED CO-SIMULATION

Distributed simulation of iteratively specified systems affords the major advantage of enabling the sharing of models and their simulation incarnations across large distances (potentially, connecting earth and space stations, for example). Usually such models are expressed in different formalisms (e.g., differential

equations, state charts, domain specific languages, etc.) so the iterative specification concept can provide a framework for such co-simulation, that is for simulating these different components in one distributed composition. The major difference between distributed simulation and its non-distributed counterpart is that information and data are encoded in messages that travel from one computer to another over a network. Such messaging requires that continuous quantities be coded into discrete packets or cells and sent discontinuously so that discrete event simulation is the most natural means to employ here. However, as mentioned, the models to be simulated cannot be restricted to discrete event form only since many, especially coming from the physical sciences, are cast in differential equation terms and may come packaged in discrete time simulators, the traditional way of simulating continuous systems.TMS2018 Chapter 20 discusses the theoretical foundation for the quantized systems to show how DEVS-based quantization provides more efficient simulation than traditional simulation of continuous systems. This forms the basis for the DEVS Bus, a distributed simulation framework for multi-formalism modeling based on DEVS component-wise co-simulation of iteratively specified coupled systems.

6 ITERATIVE SPECIFICATION MODELING EXAMPLE: LANGUAGE OF THE BRAIN

Despite growing scientific interest, we know little about how the brain is organized into circuits that function as individual processors and the language that they use to communicate and interact (Makin 2016). Neuroscience seeks to understand how such circuits lead to perception, thought, and behavior. However, currently, the gap between circuits and behavior is too wide to bridge (Carandini 2012). Neural computations that occur in individual and populations of neurons offer an intermediate level to bridge the gap. Computation supplies a common language for neural and cognitive researchers. Here, we use this approach to suggest how iterative specification modeling can support linking the language of the brain with high level cognition. Bridging the gap requires focusing on a cognitive domain of interest and showing how the functions involved can be, through stages of hierarchical decomposition, eventually implemented in neural "wet-ware." In our current state of knowledge, development of the successive stages must realistically be driven by abstractions and simplifications that enable model construction that can be related both to plausible biological constraints as well as observed data gathered at higher levels.

Table 4: Application of the modeling and simulation framework to Neurocognition.

Entity	**Application to Language of the Brain**
Source system	Greeting interactions
Behavior database	Collection of data gathered at cognitive and neuronal levels
Experimental frames	Regular expression mediation of speech perception to speech production flow
Models	Iterative system specification/DEVS formalism modeling constructs (TMS2018 Chapter 10) and DEVS Markov abstractions (Chapters 21-23)
Simulators	DEVS simulation protocol implemented in workstation integrated development environments (e.g., DEVS MS4) and/or neuromorphic hardware (e.g., Spiking Neural Network Architecture),

To illustrate how iterative system specification modeling can apply to bridging the neural-cognitive gap, let's consider the greetings interaction introduced earlier and summarized in Table 4. At the cognitive level, greetings are exchanged through natural language for which much is known about its syntax, semantics, and pragmatics. In the hierarchy of specification framework, this is at the I/O Behavior level and we need to climb to the I/O Function and State System levels to get more explanatory, mechanistic,

and eventually predictive capability. Finally, at the coupled systems level we come to neural networks that can eventually yield understanding of the language of the brain that mediates the exchanges of greetings in natural language. Let's examine each level in turn.

Although available natural language computational frameworks deal with the complexities of fully expressive conversations, here by design, we focus on the simplest of such interactions. Accordingly, we need a basis that is adequate for expressing greetings but also has potential for scaling up to more complex conversations. We identify the finite state regular expression language framework as such a basis. In brief, regular expressions are strings of symbols that beside terminals (e.g., the alphabet) contain connectors for concatenation, Boolean operations, and Kleene star (denoting iterative concatenation.) Regular expressions are descriptions of behavior at the I/O Behavior level for which there are computable realizations by finite state automata. Indeed, for every regular expression there is a finite state acceptor (that can recognize strings in its language) and a finite state generator that can generate such strings. Greetings can then be modeled as involving strings generated and recognized by such finite state systems. Therefore, we take this framework as the basis for the computations that mediate between cognitive and neural domains. However to connect with the latter, we need to identify realizations of such system specifications at the coupled system level involving neural componentry and potentially other needed cellular elements.

One possibility for transition to the coupled model level is to recognize a regular expression as a hierarchical construction based on successive application of its operators. For instance, "grumpy || cheerful & morning || cheerful & noon" is composed of concatenated letters for words, combined by the conjunction (&) and then by the disjunction (||). It could express a perception that leads to no response (if the person appears to be grumpy) or either "good morning" or "good afternoon" (if s/he appears to be cheerful.) If we have atomic models for the letters and templates for composing with the operators then we can realize such expressions by hierarchical DEVS models. More precisely, recognizing that words are abstractions of patterns of sounds over time, we actually need coupled iterative system specification models that can be computationally simulated in DEVS environments. Indeed, we can employ time segment generators (Figure 5) to model such sound patterns and connect to neural compositions that can generate and recognize such patterns. Also we must not forget that the composition templates also need to be expressed in plausible neural architectures. Currently, for example, we are exploring spike density generators (where density or spike frequency could code for letters (Chapters 23)) as well as activation and input/output ports coupled by sequential and parallel coupling patterns. Hence, these are the proposed elements of the language of the brain.

7 CONCLUSION

The iterative systems specification, introduced in the first edition of Theory of Modeling and Simulation, has been expanded in the third edition to a modeling formalism that enables models at any level of structure or behavior to be developed and transparently mapped into DEVS. This opens up numerous research avenues that include computational support of the various subclasses of iterative specification (Section 4.1) These classes represent concepts that were once separate continuous and discrete formalisms. These concepts are now unified under a single framework enabling models in diverse formalisms to be co-simulated in a single environment. The example of two-person greeting encounters and the attempt to bridge the behavior to circuits gap discussed here illustrate the transdisciplinary problems that are awaiting the wider spread application of iterative specification modeling and simulation.

REFERENCES

Barros, F.J. 2002. "Modeling and Simulation of Dynamic Structure Heterogeneous Flow Systems". *Simulation* 77(1):552–561.

Carandini, M. 2012. "From Circuits to Behavior: A Bridge Too Far?". *Nature Neuroscience* 15(4):507-509.

Makin, S. 2016. Deciphering the Language of the Brain. *Scientific American*, published online January 31. 2016. https://www.scientificamerican.com/article/deciphering-the-language/, accessed 15th Sept 2019.

Mittal, S. and B. P. Zeigler. (2014). "Modeling Attention Switching in Resource Constrained Complex Intelligent Dynamical Systems (RCIDS)." In *Proceedings of the 2012 Spring Simulation Multi-Conference*, Symposium on Theory of M&S/DEVS, Article 23. San Diego, CA: Society for Computer Simulation International.

M&SCO. 2016. M&S Reference Glossary. Defense Modelling & Simulation Coordination Office (M&SCO). https://www.msco.mil/MSReferences/Glossary/TermsDefinitionsS-W.aspx, accessed 15th Sept 2019.

Wikipedia, Regular Expressions, last modified 7th September 2019. https://en.wikipedia.org/wiki/Regular_expression, accessed 15th September 2019.

Zeigler, B. P., A. Muzy, A., and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations, 3rd Edition*. New York, NY, USA: Academic Press.

ADDITIONAL READING

Zeigler, B.P. and H. Sarjoughian. 2017. *Guide to Modeling and Simulation of System of Systems*. New York, NY, USA: Springer.

AUTHOR BIOGRAPHIES

BERNARD ZEIGLER is Professor Emeritus of Electrical and Computer Engineering at the University of Arizona, Tucson, AZ, USA and Chief Scientist of RTSync Corp., Phoenix, AZ, USA. He is internationally renowned for his seminal contributions in modeling and simulation theory. He is well known for the Discrete Event System Specification (DEVS) formalism that he invented in 1976 and which is now being used world-wide in advanced information systems. Dr. Zeigler is a Fellow of IEEE and SCS and received the INFORMS Lifetime Achievement Award. He is a co-director of the Arizona Center of Integrative Modeling and Simulation. His e-mail address is zeigler@rtsync.com.