A Theory-based Conceptual Terminology for M&S VV&A

Bernard P. Zeigler

AI and Simulation Group
Department of Electrical and Computer Engineering
University of Arizona,
Tucson, Arizona
zeigler@ece.arizona.edu

Keywords:
Modeling theory, formalism, verification, validation, layered architecture, separation of concerns

ABSTRACT: The DEVS (Discrete Event System Specification) is a sound formal Modeling and Simulation framework based on generic dynamic systems concepts. DEVS is a mathematical formalism with well defined concepts of coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and an object-oriented substrate supporting repository reuse. Despite the variance with other terminologies widely prevalent in the DoD M&S community, we propose that there are advantages for VV&A to be gained by adopting the theory-based definitions. These advantages include well-defined separation of concerns supporting distinct modeling and simulation layers that can be independently verified and reused in later combinations with minimal re-verification. The resulting divide and conquer approach can greatly simplify and accelerate VV&A leading to greater credibility at reduced effort. An example involving verification of the implementation of the DEVS/HLA environment is given to illustrate these points.

1. Introduction

The DEVS (Discrete Event System Specification) is a sound formal modeling and simulation (M&S) framework based on generic dynamic systems concepts. DEVS is a mathematical formalism with well defined concepts of coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and an object-oriented substrate supporting repository reuse. The framework has been adopted by a book intended for widespread DoD application [1] In a separate SIW tutorial, (Creating Simulations in HLA/RTI using the DEVS Modeling Framework), we explain the DEVS methodology and DEVS/HLA, an HLA-compliant modeling and simulation environment that supports high level model building using the DEVS methodology. In this paper, we propose that the separation of concerns between modeling and simulation realized in DEVS/HLA affords a workable, theory-based approach to some of the problems of VV&A arising in distributed simulation. To substantiate this claim we present the conceptual elements of the DEVS framework for M&S and show that many of the difficult-to-manage concepts of verification and validation are captured with in it. Some of the concepts, such as experimental frame, identified by the framework are related to, but are not necessarily, synonymous with, similar terminology widely adopted by the DoD M&S community[2]. Despite the variance with other terminologies, we propose that there are advantages for VV&A to be gained by adopting the theory-based definitions. These advantages include well-defined separation of concerns supporting distinct modeling and simulation layers that can be independently verified and reused in later combinations with minimal re-verification. The resulting divide and conquer approach can greatly simplify and accelerate VV&A leading to greater credibility at reduced effort.

2. Framework for Modeling and Simulation

The Framework for M&S in Figure 1 establishes entities and their relationships that are central to the M&S enterprise. Terms such as “model” and “simulator” are often loosely used in current practice but have a very sharp meanings in the framework we will discuss. Based on this framework, the basic issues and problems encountered in performing M&S activities can be better understood and coherent solutions developed Therefore, it is important to understand what is included and excluded by the definitions.
The entities of the Framework are: source system, model, simulator and experimental frame; they are related by the modeling and the simulation relationships. Each entity is formally characterized as a system at an appropriate level of specification of a generic dynamic system. Likewise each relationship is characterized as an appropriate morphism between corresponding dynamic systems. Presentation of the systems theory background is beyond the scope of this paper but can be found in [3] and is briefly reviewed in another paper in these proceedings [4]. Assuming this background, the following briefly reviews the concepts of the M&S Framework.

Figure 1. Framework Entities and Relationships

2.1 Source System

The source system 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. This data is viewed or acquired through experimental frames of interest to the modeler.

2.2 Experimental Frame

An experimental frame is a specification of the conditions under which the system is observed or experimented with. An experimental frame is the operational formulation of the objectives that motivate a modeling and simulation project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions (Figure 2).

2.3 Model

In its most general guise, a model is a system specification at any level in the behavior-structure hierarchy. However, in the traditional context of M&S, the system specification is usually at a high level, such as a set of instructions, rules, equations, or constraints for generating input/output behavior. In other words, we write a model with state transition and output generation mechanisms 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 models at a higher level specification. Models may be expressed in a variety of formalisms which may be understood as means for specifying subclasses of dynamic systems. The DEVS formalism delineates the subclass of discrete event systems and, through its representational ability, it can express the systems specified within traditional formalisms such as differential (continuous) and difference (discrete time) equations [3].
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 specification has the advantages that it has a sound mathematical foundation and has a definite semantics that everyone can understand in unambiguous fashion.

2.4 Simulator

A simulator is any computation system (such as a single processor, or a processor network, or more abstractly an algorithm), capable of executing a model to generate its behavior. The more general purpose a simulator is the greater the extent to which it can be configured to execute a variety of model types. In order of increasing capability, simulators can be:

1. dedicated to a particular model or small class of similar models
2. capable of accepting all (practical) models from a wide class, such as an application domain (e.g., communication systems)
3. restricted to models expressed in a particular modeling formalism, such as continuous differential equation models
4. capable of accepting multi-formalism models (having components from several formalism classes, such as continuous and discrete event).

DEVS/HLA is a simulation environment – simulator together with supporting development tools – that can execute DEVS models in an HLA compliant manner. The High Level Architecture (HLA) standard is being promoted by the Defense Modeling and Simulation Organization to facilitate interoperability of distributed simulation. DEVS/HLA makes a clear separation of modeling and simulation layers as will be discussed soon.

2.5 Relationships

The entities – system, experimental frame, model, simulator – take on real importance 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. Two of the most important are validity and simulator correctness.

2.6 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 our formulation, 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 input/output (I/O) relation level 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. 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 often used 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 highly detailed and valid (in some understood experimental frame). However when used this way, the assumption seems to be 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.

2.7 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. In practice, as suggested above, simulators are constructed to execute not just one model but a family of possible models. 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. As will be discussed in a moment, a homomorphism is a correspondence between simulator and model states that is preserved under transitions and outputs.

Figure 3 Base/Lumped Model Equivalence in Experimental Frame

2.8 Other Relationships

Besides the two fundamental relationships, there are others that are important for understanding modeling and simulation work. These relations have to with the ordering of models and experimental frames. We have already mentioned the homomorphism concept which is a relation between two systems described at a high level of system specification.

2.9 Base/Lumped Model Relationships

In addition, it is often critical to know whether it makes sense to apply a given experimental frame to a model. The relation that determines if such an application is possible is called applicability and its converse, is called accommodation.

Notice that validity of a model in a particular experimental frame, requires, as a precondition, that the model accommodates the frame. For useful model repositories it is critical to have an ability to ask whether there are any experimental frames that meet our current objectives and whether there are models that can accommodate this frame. Only those models have a chance of providing valid answers to our current questions.

The degree to which one experimental frame is more restrictive in its conditions than another is formulated in the derivability relation. A more restrictive frame leaves less room for experimentation or observation than one from which it is derivable. So, as illustrated in Figure 4, it is easier to find a model that is valid in a restrictive frame for a given system.

Figure 4. Illustrating Important M&S Relations.

2.10 Verification

Verification is the attempt to establish that the simulator correctness relation holds between a simulator and a model. Actually, as we have seen, the simulator may be capable of executing a whole class of models. In this case, we wish to guarantee that it can execute any of these models correctly. The basic concept in establishing correctness is that of homomorphism. When a model should go through a state sequence such as $a, b, c, d$, the simulator 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 simulator states and models states are identical – only that there is a predefined correspondence between them illustrated by the shaded connecting lines in the figure.

Figure 5 Homomorphism Concept

Now to establish that this correspondence is a homomorphism requires that whenever the model specifies a transition, such as from state $b$ to state $c$, then the simulator 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 a simulator designer can show that such a homomorphism holds then any state
trajectory in the model will be properly reproduced by the simulator.

There are two general approaches to verification:

- formal proofs of correctness
- extensive testing

Formal proofs employ mathematical and logical formalisms underpinning the concepts in the systems specification hierarchy to rigorously establish the requisite homomorphism. Unfortunately, such proofs are difficult or impossible to carry out for large, complex systems. Moreover, they may also be prone to error since ultimately humans have to understand the symbols and carry out their manipulations. On the positive side, more automated tools are becoming available to relieve some of the burden.

In the absence of once-and-for-all proofs, extensive testing must be done to assure that all conditions that could arise in simulator operation have been covered by test cases. Time and other resources limit the amount of testing that can be done. However, even though formal proofs may not be employed, the homomorphism concept still offers a framework for laying out the combinations of inputs and states that have to be tested for thorough, yet efficient, testing.

3. Separation of Concerns in DEVS/HLA

The distinction between models and simulators underlies the layered structure of the DEVS/HLA distributed simulation environment illustrated in Figure 6. At the highest layer, modelers construct their models in the DEVS formalism independently of the underlying simulation protocol and lower layers. These lower layers can be implemented in a multiplicity of ways and will correctly simulate DEVS models so long as a homomorphism such as just discussed can be verified. In the DEVS/HLA environment this layered plug-and-play capability is illustrated in the two implementations based on different languages, C++ and Java, of the HLA/RTI. While DEVS models are written in different syntax for each, their essentials remain invariant and the translation from one to the other is straightforward. Likewise the DEVS Simulation protocol is abstractly specified [3] while it is implemented in necessarily different ways to accommodate some of the fundamental differences of the C++ and Java languages such as pointers, threads, and memory management.

Verification of DEVS/HLA implementation: Problems and Solutions

To deal with the correctness of these implementations we assume that the underlying RTIs are correctly implemented with respect to the HLA specification. While verification based on formal proof is feasible, we have not yet attempted that, but instead have employed an extensive suite of test cases. The suite consists of DEVS models and is to designed to cover various independent aspects of the DEVS simulation protocol. Since the correct behaviors of the models are known, an implementation of the protocol can be shown to be incorrect if comparison of any of the generated behaviors to their known counterpart standards reveals a discrepancy. Of course passing, without error, a finite suite of tests cannot conclusively establish the correctness of an implementation. However, the comprehensive coverage of all aspects of DEVS functionality in a test suite can afford significant levels of confidence in correctness.

The foregoing testing of the first DEVS/HLA C++ implementation in C++ RTI v1.0 revealed an important problem which could have adversely impacted the expressiveness of the DEVS implementation. It was not clear at the outset whether the issue was a flaw in the HLA specification, its implementation in the current RTI, or in the implementation of DEVS itself. However, the separation of layers just described enabled us to take for granted the correct implementation of the DEVS layer -- which had been previously independently verified [5]. This enabled us to pin point the problem to the HLA layer and eventually to the HLA specification itself. A resolution was found that exploited formal properties of the DEVS formalism and efficient use of the HLA time management services. (This resolution is discussed in another paper in these proceedings [6]). Because it has been formally characterized in terms of the DEVS formalism, its simulation protocol and the HLA specification, the solution is portable to HLA-compliant implementations in other languages such as Java, as suggested in Figure 6.

1 Actually, in this case the underlying RTI is the same, viz., C++ v1.3 , the Java version difference in having a language specific API (application specific interface).
4. Conclusions

We have proposed that the separation of concerns between modeling and simulation realized in DEVS/HLA affords a workable, theory-based approach to some of the problems of VV&A arising in distributed simulation. To substantiate this claim we have illustrated, from actual experience, how the well-defined distinct modeling and simulator layers can be independently verified and reused in later combinations with minimal re-verification. The resulting divide and conquer approach can greatly simplify and accelerate VV&A leading to greater credibility at reduced effort.

5. References


BERNARD P. ZEIGLER is Professor or Electrical and Computer Engineering at the University of Arizona, Tucson. He has written several foundational books on modeling and simulation theory and methodology. He is currently leading a DARPA sponsored project on DEVS framework for HLA and predictive contracts. He is a Fellow of the IEEE.