DEVS COMPONENT-BASED M&S FRAMEWORK: AN INTRODUCTION

Bernard P. Zeigler

Arizona Center for Integrative Modeling & Simulation
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ, 85721-0104, USA
http://www.acims.arizona.edu zeigler@ece.arizona.edu

Hessam S. Sarjoughian

Arizona Center for Integrative Modeling & Simulation
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ, 85287-5406, USA
http://www.acims.arizona.edu hessam.sarjoughian@asu.edu

ABSTRACT

This tutorial describes the DEVS modeling and simulation framework and its underlying fundamental modeling concepts. We exemplify the DEVS formalism atomic and coupled models using simple, novel discrete event neurons. We discuss the hierarchical, modular composition approach derived from systems theory and show that it affords a good basis for model reusability. We conclude with an observation that models developed in the DEVS framework can be executed in either centralized/parallel/distributed computing environments without changing their dynamic characterizations and consequently their interpretations/execution.

1 FRAMEWORK FOR MODELING AND SIMULATION

The Discrete Event System Specification (DEVS) formalism provides a means of specifying a mathematical object called a system [Zeigler, et. al, 2000]. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs. Discrete event systems represent certain constellations of such parameters just as continuous systems do. For example, the inputs in discrete event systems occur at arbitrarily spaced moments, while those in continuous systems are piecewise continuous functions of time. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. Having this abstraction, it is possible to design new simulation languages with sound semantics that easier are to understand. Indeed, the DEVJAVA environment [ACIMS, 2002] is an implementation of the DEVS formalism in Java which enables the modeler to specify models directly in its terms.

1.1 Brief Review of the DEVS Concepts

Figure 1 depicts the conceptual framework underlying the DEVS formalism [Zeigler and Sarjoughian, 2001]. The modeling and simulation enterprise concerns three basic objects:

- the *real system,* in existence or proposed, which is regarded as fundamentally a source of data
- *model*, which is a set of instructions for generating data comparable to that observable in the real system. The *structure* of the model is its set of instructions. The *behavior* of the model is the set of all possible data that can be generated by faithfully executing the model instructions.
- *simulator*, which exercises the model's instructions to actually generate its behavior.
- *experimental frame,* which captures how the modeler's objectives impact on model construction, experimentation and validation. As we shall see later, in DEVJAVA experimental frames are formulated as model objects in the same manner as the models of primary interest. In this way, model/experimental frame pairs form coupled model objects with the same properties as other objects of this kind. It will become evident later, that this uniform treatment yields immediate benefits in terms of modularity and system entity structure representation.

The basic objects are related by two relations:

- *modeling relation* linking real system and model, defines how well the model represents the system or entity being modeled. In general terms a model can be considered valid if the data generated by the model agrees with the data produced by the real system in an experimental frame of interest.
- *simulation relation,* linking model and simulator, represents how faithfully the simulator is able to carry out the instructions of the model.

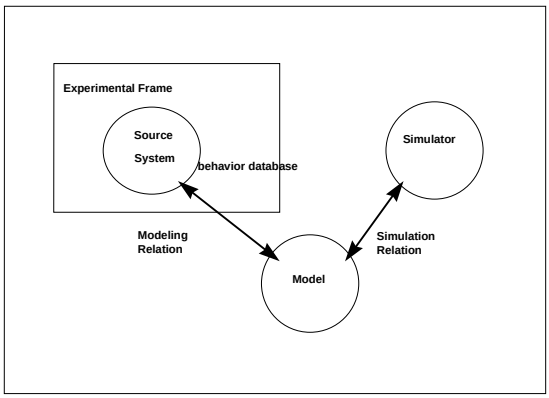

Figure 1 Basic Entities and Relations

The basic items of data produced by a system or model are *time segment*s. These time segments are mappings from intervals defined over a specified time base to values in the ranges of one or more variables. The variables can either be observed or measured. An example of a data segment is shown in Figure 2.

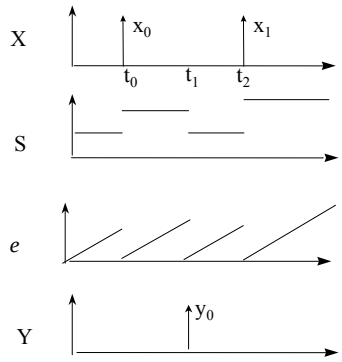

Figure 2 Discrete event time segments

The structure of a model may be expressed in a mathematical language called a *formalism*. The discrete event formalism focuses on the changes of variable values and generates time segments that are piecewise constant. Thus an event is a change in a variable value which occurs instantaneously.

In essence the formalism defines how to generate new values for variables and the times the new values should take effect. An important aspect of the formalism is that the time intervals between event occurrences are variable (in contrast to discrete time where the time step is generally a constant number).

1.2 Basic Models

In the DEVS formalism, one must specify 1) basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion.

To specify modular discrete event models requires that we adopt a different view than that fostered by traditional simulation languages. As with modular specification in general, we must view a model as possessing input and output ports through which all interaction with the environment is mediated. In the discrete event case, events determine values appearing on such ports. More specifically, when external events, arising outside the model, are received on its input ports, the model description must determine how it responds to them. Also, internal events arising within the model, change its state, as well as manifesting themselves as events on the output ports to be transmitted to other model components.

A *basic model* contains the following information:

- the set of input ports through which external events are received,
- the set of output ports through which external events are sent,
- the set of state variables and parameters: two state variables are usually present, "phase" and "sigma" (in the absence of external events the system stays in the current "phase" for the time given by "sigma"),
- the time advance function which controls the timing of internal transitions – when the "sigma" state variable is present, this function just returns the value of "sigma",
- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed,
- the external transition function which specifies how the system changes state when an input is received – the effect is to place the system in a new "phase" and "sigma" thus scheduling it for a next internal transtion; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state,
- the confluent transition function which is applied when an input is received at the same time that an internal transition is to occur – the default definition simply applies the internal transition function before

applying the external transition function to the resulting sate, and

- the output function which generates an external output just before an internal transition takes place.

2 THE DEVS FORMALISM

In this section we start with the basic DEVS formalism and discuss an example using it. We then discuss the DEVS formalism for coupled models also giving examples.

2.1 Parallel DEVS

A *Parallel Discrete Event System Specification* (*DEVS*) is a structure

$$M = \langle X,S,Y,\delta_{int},\delta_{ext},\ \delta_{con},\lambda,\ ta\rangle$$

where

X is the set of input values
S is a set of states,
Y is the set of output values
δ_{int}: $S \rightarrow S$ is the *internal transition* function

δ_{ext}: $Q \times X^b \rightarrow S$

is the *external transition* function, where
$Q = \{(s,e) \mid s \in S,\ 0 \le e \le ta(s)\}$ is the *total state* set
e is the *time elapsed* since last transition

X^b denotes the collection of bags over X (sets in which some elements may occur more than once).

δ_{con}: $Q \times X^b \rightarrow S$

is the *confuentl transition* function,

λ: $S \rightarrow Y^b$ is the output function

ta: $S \rightarrow \mathbf{R}^{+}_{0,\infty}$ is the *time advance* function

The interpretation of these elements is illustrated in Figure 3. At any time the system is in some state, s. If no external event occurs the system will stay in state s for time $ta(s)$. Notice that $ta(s)$ could be a real number as one would expect. But it can also take on the values 0 and ∞. In the first case, the stay in state s is so short that no external events can intervene – we say that s is a *transitory* state. In the second case, the system will stay in s forever unless an external event interrupts its slumber. We say that s is *a passive* state in this case. When the resting time expires, i.e., when the elapsed time, $e = ta(s)$, the system outputs the value, $\lambda(s)$, and changes to state $\delta_{int}(s)$. Note output is only possible just before internal transitions.

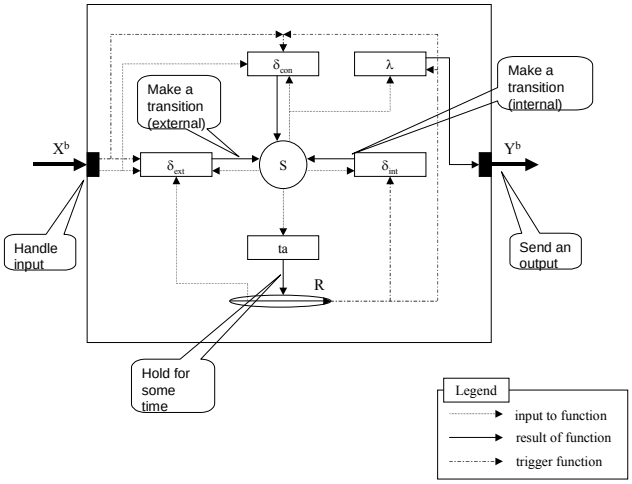

Figure 3 Parallel DEVS

If an external event $x \in X^b$ occurs before this expiration time, i.e., when the system is in total state (s, e) with $e \le ta(s)$, the system changes to state $\delta_{ext}(s,e,x)$. Thus the internal transition function dictates the system's new state when no events have occurred since the last transition. While the external transition function dictates the system's new state when an external event occurs – this state is determined by the input, x, the current state, s, and how long the system has been in this state, e, when the external event occurred. In both cases, the system is then is some new state s' with some new resting time, $ta(s')$ and the same story continues.

Note that an external event $x \in X^b$ is a bag of elements of X. This means that one or more elements can appear on input ports at the same time. This capability is needed since Parallel DEVS allows many components to generate output and send these to input ports all at the same instant of time.

Warning: There is no way to generate an output *directly* from an external input event. An output can only occur just before an internal transition. To have an external event cause an output without delay, we have it "schedule" an internal state with a hold time of zero. The relationship between external transitions, internal transitions, and outputs are as shown in Figure 3.

The above explanation of the semantics (or meaning) of a DEVS model suggests, but does not fully describe, the operation of a simulator that would execute such models to generate their behavior. Nevertheless, the behavior of a DEVS is well defined and can be depicted as we mentioned earlier in Figure 2. In that figure, the *input trajectory* is a series of events occurring at times such as t_0 and t_2. In between, such event times may be those such as t_1 which are times of internal events. The latter are noticeable on the *state trajectory* which is a step-like series of

states, which change at external and internal events (second from top). The *elapsed time trajectory* is a saw-tooth pattern depicting the flow of time in an elapsed time clock which gets reset to 0 at every event. Finally, at the bottom, the *output trajectory* depicts the output events that are produced by the output function just before applying the internal transition function at internal events. Such behaviors will be illustrated in the next chapter.

2.2 Example: Fire-Once Neuron

In a DEVS, inputs events arriving in time are handled somewhat in the manner of interrupts by the modeler-specified external transition function and result in an immediate change in state. This function determines the state transition and how long to stay in the new state. At the end of this duration, the model outputs an event determined by the output function and transits to a new state determined by the internal transition function. As an example, consider a DEVS model of the *fire-once neuron* [Zeigler, 2002]. It remains in the *receptive* state until an input arrives. The external transition function sends it to the *fire* state, where it remains for a duration given by firing-time, after which it emits a pulse and enters state *refract*. The time advance in *refract* is infinity, and it remains in *refract* upon receiving an input, so the neuron remains in *refract* forever, never able to fire again.

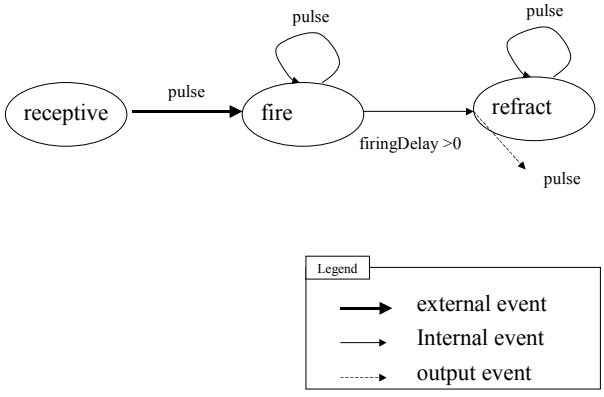

Figure 4 Fire-Once Neuron

Figure 4 illustrates the use of graphical notation to denote critical aspects of DEVS models. We'll correlate this notation with the DEVS model for the fire-once neuron, which can be expressed as follows:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

where

X is the set of input values, {pulse}
S is a set of states, {receptive, fire, refract}
Y is the set of output values, {pulse}

δ_{int}: $S \rightarrow S$ is the *internal transition* function

$$\delta_{int}(\text{fire}) = \text{refract}$$

(note: internal transitions are depicted by light transition arrows as in Figure 4)

δ_{ext}: $Q \times X^b \rightarrow S$ is the *external transition* function,

$$\delta_{ext}(\text{receptive}, e, \text{pulse}^b) = \text{fire}$$

(note: external transitions are depicted by heavy transition arrows as in Figure 4)

$$\delta_{ext}(\text{fire}, e, \text{pulse}^b) = \text{fire}$$
$$\delta_{ext}(\text{refract}, e, \text{pulse}^b) = \text{refract}$$

(note: we use the notation, pulse^b, to indicate one or more pulses arriving at the same instant. In this model, they have no combined effect greater than any one of them.)

λ: $S \rightarrow Y^b$ is the output function

$$\lambda(\text{fire}) = \{\text{pulse}\}$$

(note: outputs are depicted by dotted arrows as in Figure 4; also only a single pulse is generated.)

$\lambda(\text{receptive}) = \varphi$ (the symbol for null output, i.e., no event occurs)
$\lambda(\text{refract}) = \varphi$

ta: $S \rightarrow \mathbf{R}^+_{0,\infty}$ is the *time advance* function

ta (receptive) = ∞
ta (fire) = firingDelay
ta (refract) = ∞

Note that states that have an infinite time advance cannot generate output (since an internal transition will never occur from such a state). So we need not explicitly associate the null-event output with such states (as we have done for illustration above). In other words, the output function need only be defined for the non-passive states of a model.

2.3 Coupled Models

Basic models may be coupled in the DEVS formalism to form a *coupled model*. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction. A coupled model contains the following information:

- the set of components
- the set of input ports through which external events are received

- the set of output ports through which external events are sent

- the coupling specification consisting of:

  - the external input coupling which connects the input ports of the coupled to model to one or more of the input ports of the components – this directs inputs received by the coupled model to designated component models,

  - the external output coupling which connects output ports of components to output ports of the coupled model – thus when an output is generated by a component it may be sent to a designated output port of the coupled model and thus be transmitted externally,

  - the internal coupling which connects output ports of components to input ports of other components- when an input is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model).

Figure 5 illustrates how internal coupling directs the flow of outputs to inputs in an illustrative coupled model, AB. When outputs are generated on an output port of a component, A, they are sent at the same time instant to the input port of component, B due to a coupling of the respective output and input ports defined from A to B.

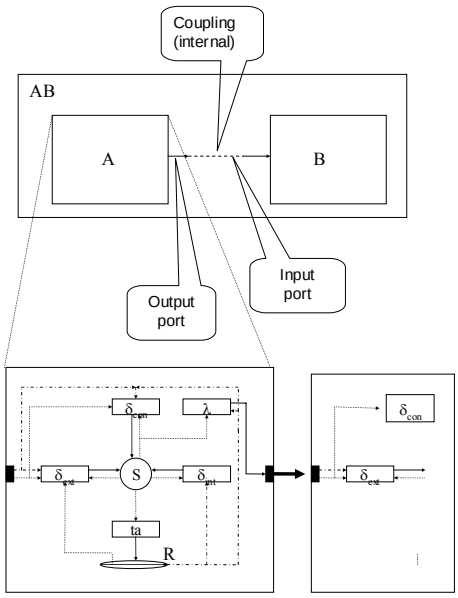

Figure 5 Coupled DEVS and the mechanism implementing the coupling specification

2.4 Neural Network Example of Coupled Model

Neural networks can be represented as coupled models within the DEVS formalism. Moreover, this possibility gives rise to a whole class of networks constructed of discrete event neuron models that includes the traditional models as a subclass. More significantly, this class includes new kinds with distinct behaviors and information processing properties. Figure 6 illustrates a small network of fire-once neurons (as in Figure 4). This kind of network can compute the shortest path in a directed graph by mapping distances of edges to an equivalent time value. This time is then assigned as the fireDelay for a neuron which represents the edge. In Figure 6, for example, a pulse emitted from the generator explores two paths concurrently to reach the final neuron (number 4). Depending of the summed firing delays along each path, a pulse emerging from one or the other will arrive earlier to the final neuron, representing the shortest path of an associated digraph. The path can be reconstructed by extending the neuron model to allow retracing the path of earliest firings. If instead of shortest paths, we request longest paths, a net of neurons that fire after their assigned fireDelays *every* time they receive a pulse will do the job. Interestingly, finding critical paths in PERT charts require such longest path computation.

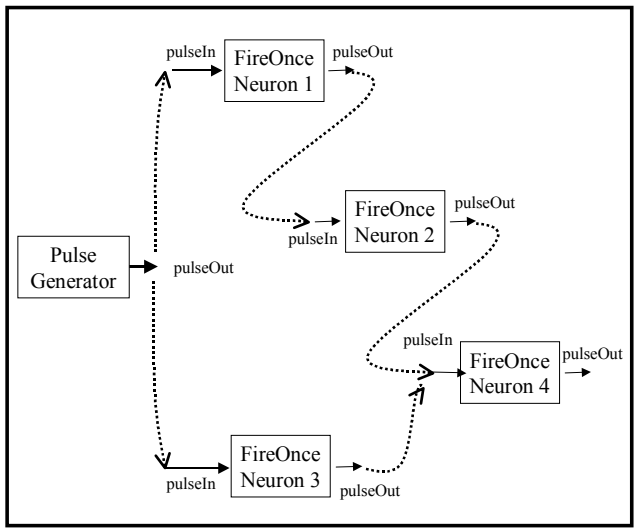

Figure 6 Neural Network Coupled Model

2.5 Hierarchical Model Construction: The DEVS composition framework

A coupled model can be expressed as an equivalent basic model in the DEVS formalism. Such a basic model can itself be employed in a larger coupled model. This shows that the formalism is closed under coupling as required for hierarchical model construction. Expressing a coupled model as an equivalent basic model captures the means by

which the components interact to yield the overall behavior.

Closure under coupling and hierarchical construction form the basis of the DEVS composition framework. The framework deals with components which are modular, i.e., are self-contained and can stand alone or be incorporated, as components into a larger system. There are two types of components: atomic models and coupled models. Atomic models are expressed directly as basic models in the DEVS formalism. Coupled models are specified by providing the set of existing components and the internal and external coupling specifications. We note that due to closure under coupling, coupled models have the same input and output port interfaces as atomic models and can be treated in the same manner as far as their external relations to other components. In particular, coupled models can become components in larger systems, just as atomic modules can, and this leads to hierarchical decomposition and construction.

Figure 7 then illustrates how by adding in a coupling specification to a set of models, we get a coupled model. By using this model as a component in a larger system with new components, and adding coupling information, we get a hierarchical coupled model.

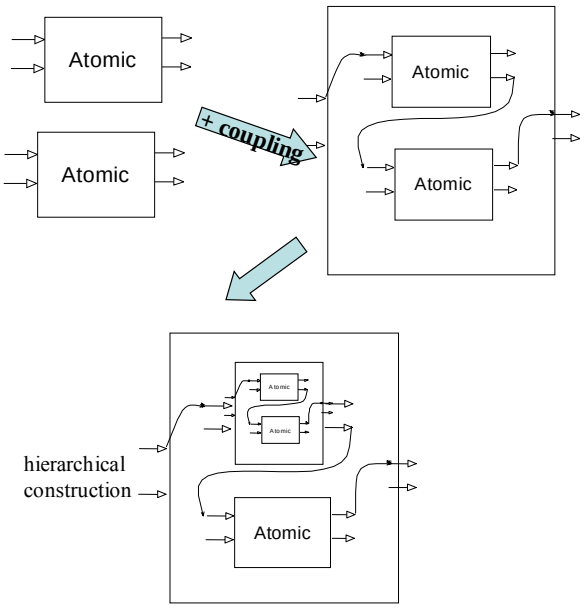

Figure 7 Coupled Modules formed via coupling and their use as components

It is important to note that DEVS models, whether atomic or coupled can standalone and go into a repository for reuse by the developers or others. In principle, the internals of any such model can be hidden (e.g., in relation to proprietary rights) – only the behavior as seen through the input/output ports needs to be communicated in a clear enough manner for others to use the component. This can foster a high degree of reuse and sharing among a growing DEVS community.

3 SCALABLE DISTRIBUTED MODELING AND SIMULATION

A direct consequence of the separation of models from simulators in the DEVS formalism is the independence of model components from alternative modes of execution – centralized, parallel, or distributed simulation execution. Model development independent of simulation execution is central to scalability and underlying computational technologies such as HLA, CORBA, or MPI. Here scalability refers to building large- or very-large scale models knowing that the suitability and usability of models is not affected by the computational resources that may be required. That is to say, model components developed for execution on a single processor can be migrated to execute on top of simulators which in turn employ middleware technologies such as CORBA and HLA. In this setting, middleware technologies provide services such as communication and time management for DEVS models to interchange input/output messages while relying on individual computational nodes to carry out each model's input, output, and state transitions. Similarly, models developed for distributed simulations may also be used for execution on a single processor. Therefore, within the DEVS framework, modelers can develop model components which may be migrated from single processor to multiprocessor and vice versa.

4 SUMMARY

The form of DEVS (discrete event system specification) discussed provides a hierarchical, modular approach to constructing reusable model components. In doing so, the DEVS formalism embodies the concepts of systems theory and modeling. Furthermore, DEVS M&S framework offers a full range of computational means to support scalability in modeling needs while ensuring models and their interpretations (behavior) remain invariant using ever more capable simulation technologies.

5 REFERENCES

ACIMS, DEVSJAVA software, http://www.acims.arizona.edu

Zeigler, B.P., (2002). The brain-machine disanalogy revisited, *BioSystems,* Vol. 64, pp. 127-140.

Zeigler, B.P., T.G. Kim, et al., (2000), Theory of Modeling and Simulation. New York, NY, Academic Press.

Zeigler, B.P., H.S. Sarjoughian. (2001). Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations, http://www.acims.arizona.edu.