DEVS Today: Recent Advances in Discrete Event-Based Information Technology

Bernard P. Zeigler
Arizona Center for Integrative Modeling & Simulation
University of Arizona
Tucson, AZ, 85721-0104, USA
http://www.acims.arizona.edu
zeigler@ece.arizona.edu

Abstract

We review the DEVS modeling and simulation framework Its fundamental concepts are discussed from the standpoint of discrete event information processing with an example drawn from recent experiments on infant cognition. We also cover the DEVS formalism's atomic and coupled models and its hierarchical, modular composition approach. Some industrial applications of the methodology are discussed in depth to highlight the formalism's utility in the development of commercial and defense information technologies

1. Introduction

Originally introduced as a formalism for discrete event modeling and simulation, the DEVS (Discrete Event System Specification) methodology has become an engine for advances within the wider area of information technology. A variety of systems theory and artificial-intelligence applications employing DEVS-based concepts were compiled by Sarjoughian and Cellier [1]. However, the wealth of research continues to expand and a second re-examination of recent advances seems in order. Particularly, this paper reviews advances from the perspective of success stories in real-world applications. It also lays the basis for claims that discrete event information processing is an important paradigm for both understanding human intelligence and developing new information technologies that can match or supercede it.

2. Primacy of discrete event perception

Our ability to process information using discrete events is a distinctly human trait and is evident from an early age – according to a very interesting new book by David and Ann Premack [2]. Discrete event processing is characterized by the ability to perceive the flow of sensory stimuli as discrete events, and to attend to both sequencing and timing of such events [6,7]. As illustrated in Figure 1, a spatial metaphor is used in which time is perceived as a continuous horizontal line flowing from left to right and events are stored as images located on the line. The *sequencing* of events is represented by locating them on the line, with earlier events to the left of recent events corresponding to how we read (e.g., in the order of occurrence of the events e1,e2,e3,e4, we have e1 as the earliest and e4 as the most recent).[1] Instead of storing absolute values of time, only interval durations are coded giving a sense of the elapsed time that processes consume. In other words, our remembered image of mowing the lawn might last a half-second while an image of a ten minute shower might last for a few milliseconds. Before discussing discrete event information processing further, we stop to review the basic DEVS formalism within a larger framework for modeling and simulation.

3. Framework for modeling and simulation

The Discrete Event System Specification (DEVS) formalism provides a means of specifying a mathematical object called a system [3,4,5]. Basically, a system has a time base, inputs, states, 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.

[1] Apparently, readers of Mandarin use a vertical line and use a top to bottom order of events [2].

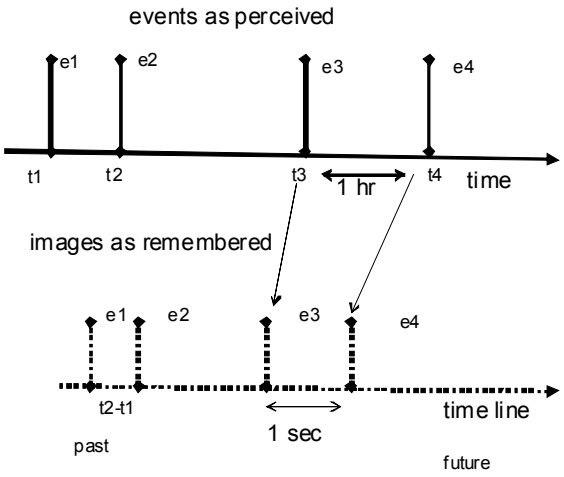

Figure 1 Representing events by their sequencing and inter-event durations.

Having this abstraction, it is possible to design new simulation environments with sound semantics that have a number of benefits to be described. Indeed, the DEVJAVA[2] environment [8] is an implementation of the DEVS formalism in Java which enables the modeler to specify models directly in its terms and derive the associated benefits.

3.1 Brief review of modeling and simulation concepts

Figure 2 depicts the conceptual framework underlying the DEVS formalism [3,4,5]. 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, or input/output pairs of time segments as described below.

- *model*, which (informally) is a set of instructions for generating data comparable to that observable in the real system[3]. The *structure* of the model is its set of instructions. The *behavior* of the model is the set of all possible input/output 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*, 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 an M&S project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions.

.

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.

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. Examples of concurrent data segments are in Figure 3.

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

[2] DEVSJAVA is the intellectual property of the Regents of the State of Arizona. Since DEVS is a public formalism, many other implementations of the concepts exist, and anyone is free to develop their own based on literature in the public domain.

[3] See [3,5] for formal characterization of the objects and relations.

(in contrast to discrete time where the time step is generally a constant number).

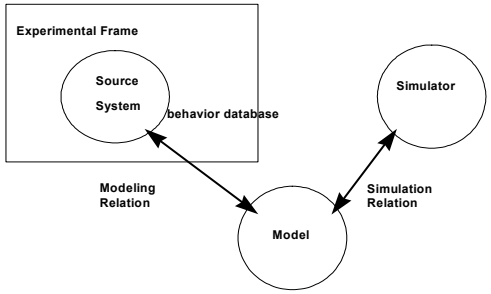

Figure 2 Basic Entities and Relations in Modeling and Simulation.

3.2 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.

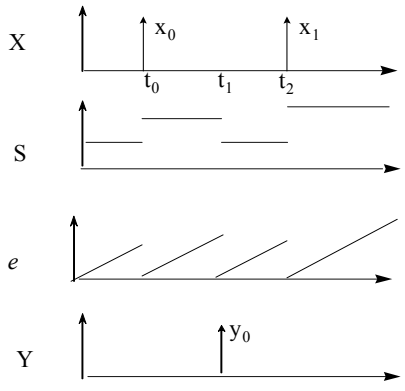

Figure 3 Four concurrent segments are shown for input X, state S, elapsed time e, and output Y. The X and Y segments are discrete event segments.

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.

The DEVS formalism uses set theory notation to concisely and rigorously state the above information. Once stated, the formalism can be used for both mathematical analysis as well as the basis for effective modeling and simulation methodology and technology development. Ashby's classic text, "Introduction to Cybernetics" provides a very readable introduction to the mathematical set theory and the systems concepts underlying the DEVS formalism. Fortunately, this historic ground-breaking work is available from http://pespmc1.vub.ac.be/books/IntroCyb.pdf. for download at no charge.

This is concisely as follows:

A *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 4. 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.

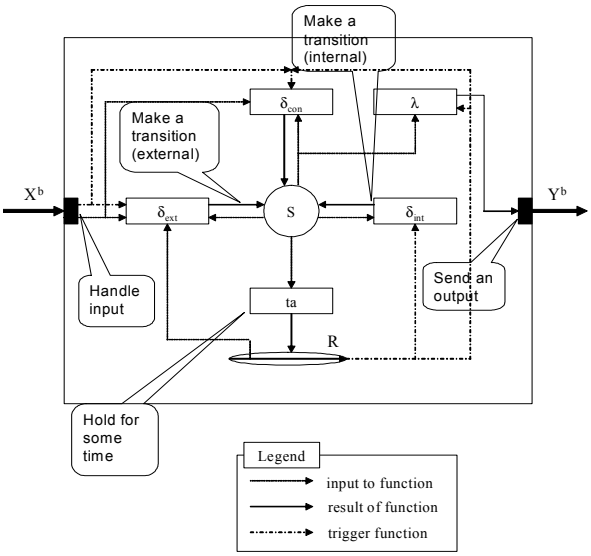

Figure 4 Portraying the operation of the DEVS formalism.

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 DEVS allows many components to generate output and send these to input ports all at the same instant of time.

4. DEVS models of mental simulation

An experimental result that supports the formulation of discrete event perception portrayed in Figure 1 is described as follows [2, page 59]: "After watching an experimenter hide two cookies (one frosted, the other plain) and then reappear eating a cookie, children consistently go to the location of the cookie that is different from the one that the experimenter is eating….When however, the cookies are wrapped in layers of tissue paper before being hidden, children no longer head for the location containing the cookie different from that being eaten by the experimenter. In notable contrast, chimpanzees fail at this problem. They continue to head for the location holding the cookie different from the one that the experimenter is eating."

The events in the experiments are illustrated attached to the time lines of Figure 5. In the first experiment, Exp 1: both children and chimpanzees assume that the reappearing cookie is one of the ones hidden Therefore they head for the other one. In the second experiment, we assume that the experimenter reappears a short time (say, 1 second) after hiding the second cookie. The results indicated that children of age four are able to recall the time (duration) it takes to unwrap cookies. Comparing it to the time available to the experimenter to do so, they conclude that the reappearing cookie could not be one of the two that were hidden. Therefore, they are indifferent to which location has a cookie. Chimpanzees apparently don't pay attention to the wrapped cookies, or if they do, can't bring information on wrapping duration to bear. Interestingly, this is a case where having more reasoning capacity, makes the outcome more uncertain not less.

Figure 6 illustrates a discrete event simulation that justifies the children's reasoning. Children are able to infer that a hidden cookie would have to be unwrapped in order to be eaten. However, given the experimenter's early reappearance this would take too much time (duration) to be possible.

The logic of a DEVS model that can be employed in the simulation is illustrated in Figure 6. The "cookie unwrapped" state has a time advance of 8 (which means that it is passive, i.e., will not undergo an internal transition). When an external event "start wrapping" occurs, the model transitions to state "wrapping" (this is part of the external transition function). The time consumed in wrapping, "time to wrap," is given by the time advance function. Once this time has elapsed, the internal transition functions dictates moving to the passive state "cookie wrapped." Notice that an external input, "start eating cookie" can only cause a transition to "eating cookie" in the "cookie unwrapped" state. So a planning simulation is set up addressing the question: how can we get from the "cookie wrapped" state to the "cookie unwrapped" state where the eating can be started. In general, such a

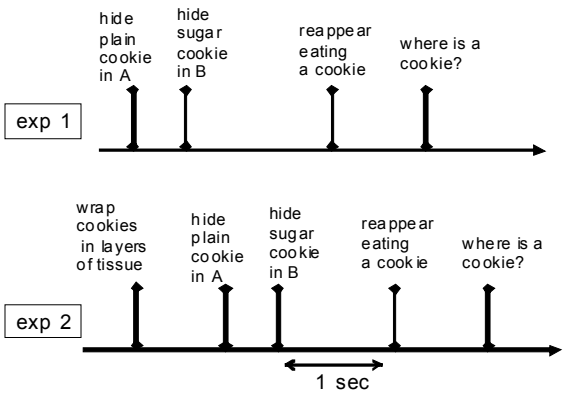

Figure 5. Time lines for experiments on discrete event perception

planning simulation will try to find a path in the state graph from the initial sate to the goal state (if there are more than one such path, an optimality criterion would be applied.) The path would reveal the sequence of external inputs and their timing to apply in order to bring the manipulated object from its current state to the goal state. In this case, the solution is apparent. As illustrated in red, a "start unwrapping" input will start the unwrapping process and one has to wait for the "time to unwrap" to elapse before the cookie will be in the "unwrapped" state ready for consumption.

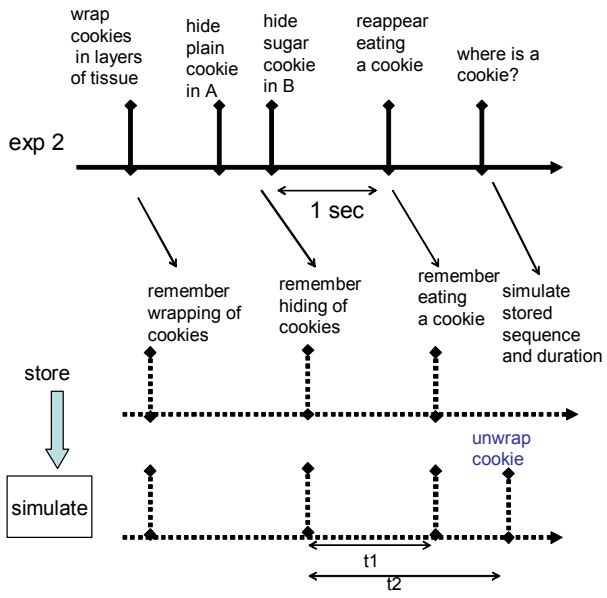

Figure 6. Illustrating how simulation is used to support temporal reasoning

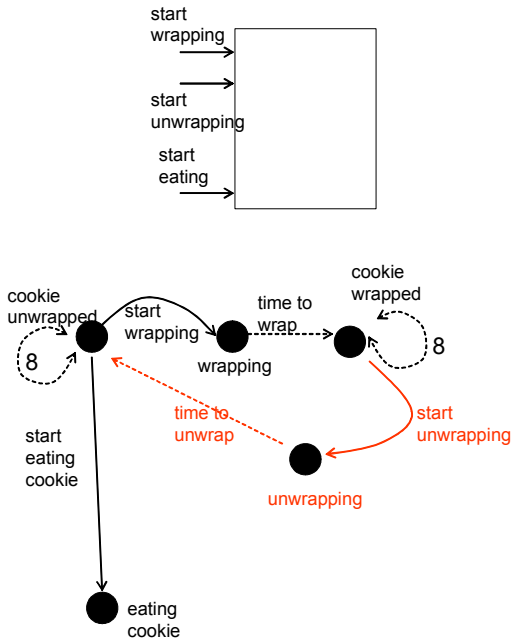

Figure 6. A DEVS model that illustrates how children reason about the wrapped cookie.

Humans and chimpanzees are able to store sequences of events and inter-event durations. Both primate species can compare stored event traces with a running event stream and with spatial analogs such as a sequence of toys laid out in some order. However, only humans can simulate stored event traces and reason with them. In this case, they can infer a necessary action, estimate its duration from past experience, compare event traces and infer their inconsistency based on incompatible durations (Figure 6). Humans, but not chimpanzees, can virtually reconstruct the past and can imagine the future. Therefore they can plan—which means they can try out alternative courses of action against a model of reality and choose what seems to be the best action to take.

5. DEVS perception-based control: an industrial success story

While our temporal information processing seems strongly founded on discrete-event principles, humans are still quite limited when it comes to the perception and control of technological phenomena that were not part of our evolutionary experience. So the question arises whether the DEVS formalism can help in developing intelligent systems that employ the discrete-event processing paradigm to control complex dynamical industrial processes.

The answer is provided in the affirmative, by Sachem, an extensive large-scale real-time system that uses DEVS as an overall framework for knowledge-based control of steel production [9]. Sachem was initially developed to monitor and to diagnose the blast furnaces of Usinor, a company in the Arcelor group, the world's largest producer of steel products. The original objective of Usinor was to save up to 1 euro per ton of pig iron [9]. Currently, Sachem saves approx. 1.6 euro per ton. and with six blast furnaces equipped with a Sachem system, Arcelor saves millions of euros annually. One the main hurdles overcome by such a system is the acquisition and the representation of the perception knowledge of the experts who monitor and diagnose the blast furnace. Such perception knowledge is sub-symbolic, meaning that experts are unable to use words to describe their own cognitive processes. Furthermore, such knowledge concerns temporal and spatial evolution of processes and humans have difficulty describing such dynamic processes with the needed fidelity. It is therefore very difficult to model

the reasoning of a control process expert.[4] Conventional artificial intelligence approaches are inadequate since they do not recognize the importance of temporal perception and discrete event processing in dealing with complex dynamical systems. The lack of mathematical models[5] of the blast furnace dynamic also inhibits subjecting it to conventional theories of process control.

The basic hypothesis underlying Sachem's design is that experts reason mainly directly on state transitions rather than on the state of the process. Because state transitions can be represented by discrete events [6,11], the knowledge to control the blast furnace can be represented with propositions about discrete events and temporal windows that constrain the time of the events

As illustrated in Figure 7, the basis of Sachem's DEVS-based perception is the recognition or successively more abstract classes of discrete events, called signal events, signal phenomena and process phenomena. The discrete event paradigm allows the design of this multi-layered self-similar discrete event abstraction process. At the lowest level, quantization [5] is employed to generate discrete event segments from continuous trajectories coming from thousands of spatially arrayed temperature, pressure and other sensors. Sachem continuously analyzes the flow of discrete events through the three levels of abstraction. The transition from one level of abstraction to the next is based on a similar event recognition process called signature detection. Since the same discrete event processing underlies each level of abstraction, a unified

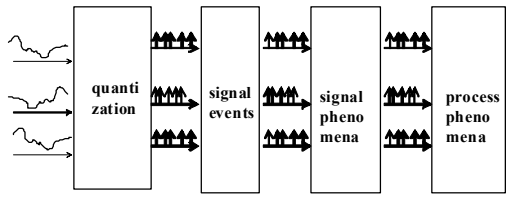

Figure 7. The continuous flow of Sachem's DEVS-based perception events

architecture can be developed which supplies essentially the same services for signature detection and reasoning at the different abstraction levels. The perception function of Sachem outputs its process phenomena events to a diagnosis function that analyzes the continuous flow of phenomena in order to recognize dangerous behaviors, to identify the hypothetical causes that explain the recognized behavior, and to issue warnings and advice to human operators on undesirable states that the process that could potentially reach in the future.

One of the advantages of the discrete event paradigm is *compactness*. For example, one year of blast furnace generates approximately 30,000 process phenomena. This results from the discrete event abstraction of a database containing the blast furnace raw data that is 50,000 this size. As a consequence, for the first time in the history of blast furnace study, it is possible to analyze the behavior of blast furnaces over years at such a high level of detail.

The success that Sachem has achieved is the result of a sustained effort with substantial investments in human and software-intensive system resources. A conceptual model of the knowledge specifies the entire system. This model contains 25,000 objects for 33 goals, 27 tasks, 75 inference structures, 3200 concepts and 2000 relations. The Sachem knowledge bases contain more than 1060 classes of objects, 1100 first order logic rules and 140 event report types. The total software volume represents approximately 400,000 lines of code. This represents 14 man-years of work for a team of 6 knowledge engineers and 12 experts during a period of 3 years.

Sachem developers are currently researching an approach to assist the expert in discovering signatures at the process phenomena abstraction level. Their approach is based on a Markov process representation of a discrete event flow. Integrated in a Java environment called the "ELP Lab", a set of tools has been developed to compute correlations between the beginnings of process phenomena. The ELP language is a high-level knowledge representation language of signatures [10]. The ELP signatures are operationalized using the DEVS formalism: the event reports are translated into DEVS models that a DEVS simulator uses in order to recognize the discrete events sequences that satisfy the signatures. The goal is to develop methods to infer an ELP signature from a set of significant sequences.

[4] Another important characteristic is the very long period (5-10 years) that is needed to learn to control the production of blast furnace

[5] One of the main characteristics of a blast furnace is the lack of a mathematical model of its internal dynamics due to the complexity of the physical and chemical interactions among gas, solids and liquids and the extreme conditions that prevail inside the blast furnace: very high temperatures, acidity, dust, high pressure, etc.

6. Coupled and Hierarchical DEVS 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 (Figure 8). This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction (Figure 9). A coupled model contains the following information:

- the set of components
- the set of input ports through which external events are received – ports are important for routing as specified by the coupling specification (below).
- 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).

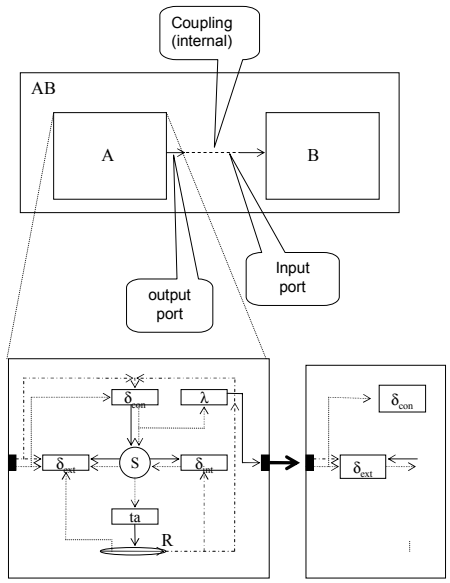

Figure 8. Coupled DEVS and the mechanism implementing the coupling specification

Figure 8 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.

6.1 Hierarchical Models

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 9 a) 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. As illustrated in Figure 9 b), the models in a hierarchical coupled model are represented in a hierarchical tree, where the leaves represent *Atomic Models* and the interior nodes represented *Coupled Models*.

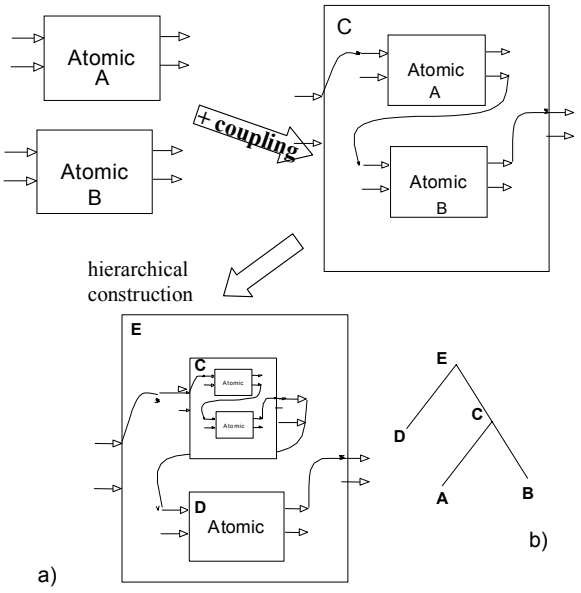

Figure 9. Coupled Modules formed via coupling and their use as components

7. DEVS-based system-of-systems simulation: a re-usability success story

It is important to note that DEVS models, whether atomic or coupled can stand alone 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.

The Advanced Simulation Center (ASC), under Steve Hall, has a long history of M&S projects with significant legacy models and simulators[12,13,14]. It is oriented toward increasing emphasis on collaboration between different parts of Lockheed Martin and with other teammates, responding to the increasing interest in performing effectiveness analysis as well as performance analysis, and increasing focus on system-of-systems solutions. For nearly a decade, ASC has adopted a DEVS-based "managed model approach," which starts from an architectural framework for constructing simulation models that supports continued cumulative development and reuse The architectural framework for organizing models is supplied by the SES (System Entity Structure [4,5]), a generalization of the hierarchical tree structure in Figure 9b). The SES specifies a well-defined structure for model reuse, maintains kind-of, part-of, multiplicity relationships, and constraints on model compatibility. Models developed are highly parameterized and are network reusable – its system-of-systems simulations federate with other joint simulation systems using HLA (High Level Architecture)[14]. ASC has a collaborative development environment, strong support for third-party model 'wrapping', and employs object-oriented development of models in DEVS/C++, an environment co-developed by ASC and the University of Arizona. Under its managed modeling paradigm, ASC has made concerted efforts to develop its models with a view to their potential reuse as components for subsequent projects. The following table indicates the success that this approach has achieved for reusability. It shows the original project in which a model was developed and the new projects for which it was modified for re-use.

A more dramatic depiction of re-usability success is illustrated by the incidence matrix in Figure 10. The matrix shows that most models have been reused

Table 1: Listing DEVS models developed by ASC over an 8 year period, the original project of development and some of the subsequent projects in which the model was reused as a component.

Acronym	Model Component	Original Project Development	Subsequent Project Re-use and modification
RAD	Radar Model	Sea Shadow	Arsenal, JCTN,
IR	IR Sensor Model	SBL	Space Discrimination
MIS	Missile Model	Arsenal	Missile Defense
LAS	Laser Model	SBL	Missile Defense, ABL
Comm.	Communications Model	MUSIC	JCTN, CMT
CC	Command Control Model	Arsenal	All projects
EARTH	Earth & Terrain Model	IRAD	CMT
WC	Weather and Cloud Model	CMT	ABL
WH	Waypoint & Heading	IRAD	CMT
OP	Orbital Propagate	SBL	Space Discrimation, CMT, NMD, MSP
BT	Ballistic Trajectory Model	Arsenal	SBL, NMD

several times after their initial development and that the modifications required (as shown in Table 1) are relatively rare. This ability to re-use models as components reliably without much further modification has provided ASC with increased capability to rapidly respond to new projects and their requirements. In fact, ASC claims to be leaving other competitive system-of-systems environments that employ monolithic (non-hierarchical, modular) development further and further behind as time goes by.

7.1 How a DEVS simulator works

The DEVS formalism has an associated well-defined concept of simulation engine to execute models and generate their behavior. A coupled model in DEVS consists of component models and a coupling specification that tells how outputs of components are routed as inputs to other components. The simulator for a coupled model is illustrated in Figure 11. It consists of a coordinator that has access to the coupled model specification as well as simulators for each of the model components. The coordinator performs the time management and controls the message exchange among simulators in accordance with the coupled model specification. The simulators respond to commands and queries from the coordinator by referencing the

Project / Model	Critical Mobile Target	Global Positioning System III	Arsenal Ship	Coast Guard Deep Water	Space Operations Vehicle	Common Aero Vehicle	Joint Composite Tracking Network	Integrated System Center	Space Based Laser	Space Based Discrimination	Missile Defense (Theater / National)
RAD	X		X	X	X	X	X				X
IR	X				X		X	X	X	X	X
MIS			X				X	X	X		X
LAS								X	X	X	X
Comm	X			X		X	X	X			X
CC	X		X								X
Earth	X	X	X		X						X
WC	X										X
WH	X	X	X	X	X		X				X
OP	X	X			X			X	X	X	X
BT			X		X	X				X	X

Figure 10. An incidence matrix showing the reuse of models as components in projects – an X indicates the model (row) was reused in a project (column).

specifications of their assigned models. The simulation protocol works for any model expressed in the DEVS formalism. It is an algorithm that has different realizations that allow models to be executed on a single host and on networked computers where the coordinator and component simulators are distributed among hosts. By replacing the simulators and their models at the leaves of Figure 11 by coordinators and their coupled models, we get a picture of how the DEVS protocol works for hierarchical coupled models.

Recently, ASC re-implemented the underlying simulation engine so as to achieve significantly reduced execution run times [13]. They could do this without disrupting the models due to the strict modular separation of model and simulator, afforded by the M&S framework within which DEVS resides.

We'll use the new ASC DEVS/C++ implementation to illustrate how an implementation of the DEVS simulation protocol works.

In DEVS/C++ Atomic models have the following methods to represent the DEVS formalism's functions:

- *delta-internal*: the internal transition function
- *delta-external*: the external transition function
- *delta-confluent*: the confluent transition function
- *output*: the output function
- *time advance:* the time advance function

Recall that the models in a hierarchical compostion are represented in a hierarchical tree (Figure 9). The role of the hierarchical tree and the couplings between pairs of models in the tree can be described as follows:

The models have *inports* through which input is provided. They also have *outports* that hold the aggregated output of the model, which will be drained when the output function of the model is exercised. A model is said to become *imminent* when a certain sleep has been completed: this results in the execution of the delta-internal function. The sleep is determined dynamically by the model through the time advance function. Coupled models contain other models (coupled or atomic) and this defines the natural parent-child relationship in the tree among the models. Atomic models, as their names imply, do not contain other models. An outport of a model can be connected to any subset of the inports of its siblings in the model tree, or to an outport of its parent coupled model. An inport of a model can also receive a connection from an inport of its parent.

The DEVS/C++ simulator maintains adjacency lists for ports and parent model pointers. In order to propagate messages, it precomputes the mail destinations for each port, dynamically and selectively updating these precomputed destination and origin sets when links, ports, or models are added to or deleted from the simulation. For example, when an outport is connected, it first calculates all the reachable inports on atomic models (the leafset) and on coupled models (the nonleafset). Likewise, when an inport is connected, similar reverse reachable sets are computed. The advantage of this obviously lies in the fact that these reachability calculations are done only when models (or ports or links) are added or deleted dynamically, and only in the part of the model tree that changed, thus adding to the efficiency of the implementation. In the following we distinguish between leafset elements and atomic models they correspond to. The leafset elements contain various pointers and slots such as the next event time and last event time that are not kept in the models themselves (for implementation independence).

A general-purpose templated priority queue for storing leafset elements by various keys, which could be a name, an next event time or other convenient index such as class name. An AVL tree structure is used to order leafset elements by their next event times: if two models have the same next event times (up to a time granule equivalence), tie breaking uses the pointer value. A reverse pointer in the element points back to its location in the priority queue makes searches efficient.

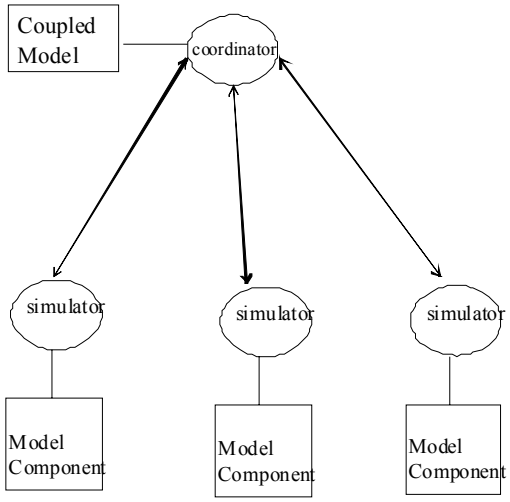

Figure 11. The DEVS Simulation Protocol

The simulation cycle is as follows:

1. Advance the clock to the small next event time in the priority queue .
2. Collect all the leafset elements that are first in the priority into a set I (representing the imminent models, i.e., those that have the smallest next event time, up to the time granule equivalence.)
3. Execute the output functions of the models pointed to by this set, and propagate the output messages directly to the inports in the leafset of the outports that produced the output. These messages are collected into the input message bags for the models in the receptive leafset.
4. Collect into a set M, all the leafset elements representing models that have non-empty bags waiting on their imports.
5. For all the elements in I intersect M (the confluent set), execute the associated model's delta-confluent function, feeding it the input bag and the elapsed time computed from the clock time and the last event time.
6. For the remaining imminents in I, execute the delta-internal function of the associated model.
7. For the remaining models in M, execute the associated model's delta-external function, feeding it the input bag and the elapsed time computed from the clock time and the last event time.

After steps 5), 6) and 7), reinsert the leafset element into the priority queue after updating its next event time obtained from the time advance function of its associated model.

Steps 1) through 7) are repeated until no models are imminent or a termination condition, such as exceeding a time or a number of iterations, is satisfied.

8. Separation of model, simulator and experimental frames: re-use in distributed simulation

We've already seen one advantage of modularity in the separation of model and simulator – the ability to change the underlying simulation engine without any change in the models themselves. There are a number of other advantages, one of which is the support of component re-usability in distributed simulation.

An experimental frame specification consists of 4 major subsections:

- *input stimuli*: specification of the class of admissible input time-dependent stimuli. This is the class of input time segments from which individual samples will be drawn and injected into the model or system under test for particular experiments.
- *control*: specification of the conditions under which an the model or system will be initialized, continued under examination, and terminated.
- *metrics*: specification of the data summarization functions and the measures to be employed to provide quantitative or qualitative measures of the input/output behavior of the model. Examples of such metrics are performance indices, goodness of fit criteria, and error accuracy bounds.
- *analysis*: specification of means by which the results of data collection in the frame will be analyzed to arrived at final conclusions. The data collected in a frame consists of pairs of input/output time segments.

When an experimental frame is realized as a system to interact with the model or system under test the 4 specifications become components of the driving system. For example, the class of input stimuli can be implemented by a generator of output time segments or alternatively, filtered from a larger set of stored or online segments. .

The HLA (High Level Architecture) [14] Federation Development and Execution Process (FEDEP) model proposes six steps in an "iterative waterfall software process" to support development of HLA-compliant simulations. The intent of HLA is to enable the construction of compositions, called federations, of existing simulations, called federates. Although, HLA and the FEDEP process provide a protocol and a methodology for such construction, the primary emphasis is on creating the right interfaces for sharing of data among federates rather than on assuring that the federates interact functionally, as models or as test environments, in the intended manner. A major problem, as suggested earlier, is that neither the HLA nor FEDEP can assure the encapsulated models, simulators, and experimentation constituents of federates match up properly for the intended application.

Figure 12 a) illustrates the inherent difficulties encountered when trying to develop testing infrastructures in the absence of separation of experimentation, model development and simulation execution concerns. Even assuming the most advanced middleware, incompatibilities at any of the three levels make it difficult for the monolithic boxes in which they reside to play meaningfully with each other. For example, the experimental frames might be incompatible because they were developed for different objectives and/or assuming different levels of resolution. Or the simulators might not be compatible because one uses event-based time advance and the other uses fixed time stepping.

In contrast, Figure 12 b) suggests an approach that allows more expeditious plug-and-play. Here, both models and frames are expressed as dynamic elements within a single formalism. The DEVS formalism supports composition of such elements into model-frame pairs and as we have seen, a standard simulation protocol supports efficient execution over existing middleware.

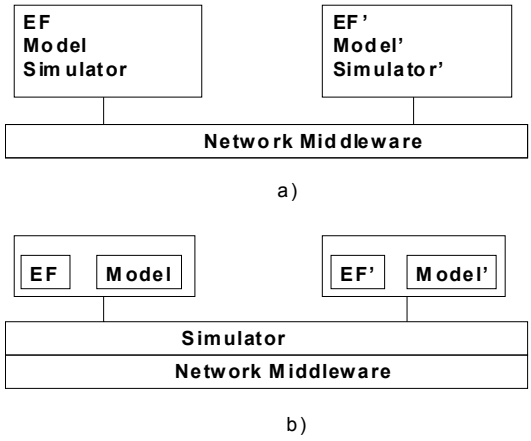

Figure 12. a) The difficulty encountered when trying to compose monolithic simulations. b) the distributed architecture for separating frame, model, and simulator.

In this case, the common use of the formalism in the simulator obviates simulation incompatibility as an issue. Further, by easily tearing the model-frame pairs apart, detecting incompatibilities at the frame and model levels can be reduced to smaller, more manageable consistency tests. For example, we can check frame consistency using the derivability relation [4]. A necessary condition for the models to form a useful composition is that their frames are derivable from (in the sense of [4]) the frame that characterizes the objectives we want to achieve with the proposed federation.

Separation of frames from models and simulators allows development of experimental frame *commodities* in both component and composite units to enable more automated and reusable test generators and the application of more powerful analytic, summarization, and visualization capabilities.

9. Endowing DEVS with more intelligence

Soft computing is a conglomeration of tools for intelligence stemming from approximate reasoning (fuzzy logic), learning (neural network, stochastic learning automaton) and optimization (genetic algorithms, genetic programming). DEVS provides a robust and generic environment for modeling and simulation applications employing single workstation, distributed, and real-time platforms. Fusing these two concepts together yields what is called IDEVS (Intelligent DEVS), an extension that is available within a virtual laboratory, called V-Lab [15]. The V-Lab environment builds on the three layers of Figure 12a). Its 3rd layer models are expressed in DEVSJAVA while an additional top layer supports distributed agent based simulation. Using the DEVS environment, V-Lab defines an appropriate structure in which to organize the elements of DEVS for a distributed agent based modeling and simulation. It separates the main components into different categories and defines the logical structure in which they communicate. It also provides the critical objects needed to control the flow of time and the flow of messages. Developed by the Autonomous Control Engineering (ACE) Center, University of New Mexico, under NASA sponsorship, V-Lab and IDEVS are targeted to development of small robotic rovers that can form cooperative teams in exploration of new terrains such as found on Mars.

V-Lab demonstrates the ability of the DEVS formalism to support the design of soft-computing intelligent behavior mechanisms that control continuous motion of vehicles over challenging terrain. V-Lab also demonstrates the benefits of working within the DEVS framework for hierarchical, modular composition and separation of model and simulation layers to provide reusable components for distributed agent simulation.

10. Moving ahead

DEVS is a framework rather than a particular technique, method or technology. As a consequence, real world applications of advanced DEVS-based information technology systems take considerable time and effort (measured in tens of person-years) to achieve. Indeed, as they develop, intelligent systems, human and artificial, must deal with a plethora of special circumstances and an exponentially growing number logic conditions. What we can hope to achieve with the DEVS formalism is a framework that scales up as such complexity increases. What are some of the ways in which further advances are needed to enable this scalability?

- Enhancing DEVS perception-based processing with the use of the soft-computing elements found within IDEVS as supported by VLab.

- Tapping the biological brain for more of its efficiency features – e.g., its ability to allocate resources (attention) to processing elements that

contribute to current goals or are currently active while not blocking out entirely less active alternatives.

- Incorporating the separation between model, experimental frame and simulators within a DEVS-based standard to enable sharing and re-use of components within broad communities of developers.

- Exploiting model continuity, the ability to transition core model specification through the different design stages of system development. DEVS model continuity allows distributed real-time systems to be designed, analyzed, and tested through DEVS-based modeling and simulation studies and then, by replacing simulators with real-time execution engine equivalents, migrated with minimal additional effort to be executed in the real-time distributed application.

There are some of the directions that are the subject of current DEVS research. Besides the collection of DEVS-related works described in [1], the reader may consult the special Issue of Transaction of Society for Modeling and Simulation International (SCS) on recent advances in DEVS Methodology. The proceedings of the recent Summer Computer Simulation Conference 2003 also contains a substantial collection of DEVS-related research.

11. Acknowledgements

This research has been supported by NSF Grant No. DMI-0122227, "Discrete Event System Specification (DEVS) as a Formal Modeling and Simulation Framework for Scaleable Enterprise Design."

12. References

1. Hessam S. Sarjoughian and Francois E. Cellier (Editors), *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*, Springer-Verlag, NY. 2001.

2. David Premack, Ann J. Premack, Original *Intelligence: The Architecture Of The Human Mind*, McGraw Hill, 2003.

3. Zeigler B.P. *Theory of modeling and simulation*. John Wiley Editor, New York, 1976.

4. Zeigler B.P.*Multifaceted Modeling and Discrete Event Simulation*. Academic Press, London UK, 1984.

5. Zeigler, B.P., T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd Edition, Academic Press. New York, NY, 2000.

6. Zeigler, B.P.," Discrete Event Abstraction: An Emerging Paradigm For Modeling Complex Adaptive Systems", (Ed. L. Booker) *Advances in Adaptive Systems*, Sante Fe Institute/Oxford Press, Oxford (in press)

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

8. Zeigler, B.P. and H.S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA* www.acims.arizona.edu 2001

9. Le Goc, M. and C. Frydman, "SACHEM, a Real Time Intelligent Diagnosis System based on the Discrete Event Paradigm", submitted to *Transaction of Society for Modeling and Simulation International* (SCS), 2003.

10. Frydman C., M. Le Goc, L. Torres and N. Giambiasi, "Knowledge-Based diagnosis in Sachem using DEVS models", Special Issue of Transaction of Society for Modeling and Simulation International (SCS) on Recent Advances in DEVS Methodology, Tag Gon Kim Ed.,Vol**. 1**8, N°3, 2001, pp147-158.

11. Giambiasi N., Escude B., and S. Ghosh, "Generalized Discrete Event Specifications: G-DEVS Coupled Models", Congress SCI2000, Orlando, USA, July 2000.

12. Steven B. Hall *Report to the National Research Council's Committee On Modeling And Simulation Enhancement For 21st Century Manufacturing And Acquisition*, November, 2000.

13. Steven B. Hall, Shankar M. Venkatesan, Donald B. Wood, "A Faster Implementation of DEVS in the Joint MEASURE Simulation Environment", in *Proc. of Summer Computer Simulation Conference*, Montreal, July 2003.

14. Zeigler, B.P, Steven B. Hall. and H.S. Sarjoughian "Exploiting HLA and DEVS to Promote Interoperability and Reuse in Lockheed's Corporate Environment", *Simulation*, Vol 73**, No.** 4, November 1999, pp. 288-295.

15. Osery, A. L, M. Jamshidi, et al, "V-Lab . – A Virtual Laboratory For Autonomous Agents", *IEEE Trans. SMC,* Vol 32, No. 8, pp791-803, Dec, 2003