Discrete Event Abstraction: An Emerging Paradigm For Modeling Complex Adaptive Systems

Bernard P. Zeigler
zeigler@ece.arizona.edu
Arizona Center for Integrative Modeling and Simulation
ECE Department
The University of Arizona
Tucson, AZ 86715
web site: www.acims.arizona.edu

To appear in "Adaptation and Evolution" (Festschrift for John H. Holland) to be published by Santa Fe Institute, Oxford Press, England.

ABSTRACT

Computer modeling and simulation is recognized by John Holland and many others as the central tool with which to experiment on complex adaptive systems (CAS). Less well recognized is that in the last thirty years, advances in the theory of modeling and simulation have crystallized a new class of models suitable for the computational requirements of CAS studies. This article discusses the abstractions underlying the DEVS formalism, a system theoretic characterization of discrete event simulations, that has been widely adopted in recent years. Abstraction of events and time intervals from a continuous data stream is shown to carry information that can be efficiently employed, not only in simulation, but also in accounting for the real world constraints that shape the information processes within CAS. Indeed, an important paradigm is emerging in which discrete event abstraction is recognized as fundamental to modeling CAS phenomena at various levels of organization. Discrete event models of neurons, neural processing architectures, and "fast frugal" bounded rational decision making and shortest path solvers are discussed as examples. Such models capture ideas that are coming from various disparate directions and offer evidence that a new modeling and simulation paradigm is emerging.

Introduction

John Holland's views on modeling and simulation and why these activities are needed to make progress in understanding complex adaptive systems (CAS) are well summarized in his article in Daedalus. [1]With examples such as economies, ecologies, immune systems, developing embryos, and brains, he characterizes CAS as distributed many-ruled organizations with little or nothing in the way of central control. The aggregate behavior that emerges is too hard to predict with thought experiments or conventional mathematics with the consequence that computer simulation is an indispensable tool for study of CAS. Because many rules can be active simultaneously, massively parallel computers hold the promise of providing fast enough execution so that humans can interact with a simulated CAS much in the same way that pilots interact with flight training simulations. Mathematics and theory are not to be thrown to the wind however, since relevant pieces of mathematics can provide insight and put some bounds on what can be expected to emerge from a simulation. Indeed, the classical theory-experiment cycle, well established for physical science, is the ideal to be sought, with the computer taking the role of mother Nature in providing the outcomes of experiments.[1]

Modeling and simulation enter in yet another way in CAS study. Holland asserts that the fundamental attribute of CAS is their use of internal models to anticipate the future, basing current actions on expected outcomes. It is important to understand how such systems use internal models since so much of their behavior stems from anticipations based on them. So how can CAS build and use internal models? Some anticipatory models are no more than rules (follow the chemical gradient to food). Some are clearly more sophisticated, such as CAS models that researchers (themselves CAS) build. What determines the type of model that a CAS can build and use? Are there some better ways to build and use CAS models than others? As a student of Holland's, who has spent the past 30 years of his professional life studying modeling and simulation, I am writing this tribute to him, with the belief that there are indeed some insights to be offered to the study of CAS and the next generations of CAS investigators.

Stated baldly and briefly – I suggest that discrete event models offer the right abstraction for capturing CAS structure and behavior. And if this is true, then the internal models that CAS build and use are also best characterized as discrete event abstractions.

Pinker [2]provides our basic perspective. Any CAS must live within the constraints imposed by the underlying material substrate on its information processing. Information has costs:

- *Space*: the hardware to hold it.
- *Time*: life is a series of deadlines. If too much information is accessible, processing it may take too much time. A CAS must put survival before sagacity.
- *Resources*: information processing and storage require energy which is always in limited supply.

The implication is well stated by Pinker: “any intelligent agent incarnated in matter, working in real time, and subject to the laws of thermodynamics must be restricted in its access to information. Only the information that is relevant should be allowed in. This does not mean that the agent should wear blinkers or become an amnesiac. Information that is irrelevant at one time for one purpose might be relevant at another time for another purpose. So information must be *routed.* Information that is always irrelevant to a kind of computation should be permanently sealed off from it. Information that is sometimes relevant and sometimes irrelevant should be accessible to a computation when it is relevant, in so far as that it can be predicted in advance.”. [2]

Why is discrete event abstraction right for modeling and simulation of CAS? It is the only formalism that can express all the constraints on information processing that are essential in understanding why CAS do what they do. For example, of the many ways that human minds could have evolved to do spatial vision, only relatively few can work within the constraints on information processing. So including these constraints in developing models for human spatial vision is indispensable to coming up with valid models[2]. Specifically, while other formalisms allow representation of space and resources, only discrete event models offer the additional ability to explicitly and flexibly express time and its essential constraints on CAS behavior and structure.

Discrete Event Modeling and Simulation

Discrete event models can be distinguished along at least two dimensions from traditional dynamic system models – how they treat passage of time (stepped vs event-driven) and how they treat coordination of component elements (synchronous vs asynchronous). Recent event-based approaches enable more realistic

[1] Reliance on simulation is not necessarily a sina qua non of CAS. Astrophyisics is in the same boat – high performance computer experiments are the basic mode of the study of stellar evolution and supernova explosions.

representation of loosely coordinated semi-autonomous processes, while traditional models such as differential equations and cellular automata tend to impose strict global coordination on such components. Event-based simulation is inherently efficient since it concentrates processing attention on events – significant changes in states that are relatively rare in space and time – rather than continually processing every component at every time step. [2]

DEVS Formalism

The DEVS (Discrete Event Systems Specification) formalism [4]provides a way of expressing discrete event models and a basis for an open distributed simulation environment. [5]DEVS is universal for discrete event dynamic systems and is capable of representing a wide class of other dynamic systems. Universality for discrete event systems is defined as the ability to represent the behavior of any discrete event model where "represent" and "behavior" are appropriately defined. Concerning other dynamic system classes, DEVS can exactly simulate discrete time systems such as cellular automata and approximate, as closely as desired, differential equation systems. This theory is presented in. [5]It also supports hierarchical modular construction and composition methodology.. [6] This bottom-up methodology keeps incremental complexity bounded and permits stage-wise verification since each coupled model "build" can be independently tested.

Discrete Event Abstractions

An abstraction is a formalism that attempts to capture the essence of a complex phenomenon relative to a set of behaviors of interest to a modeler. A discrete event abstraction represents dynamic systems through two basic elements: discretely occurring *events* and the *time intervals* that separate them (Figure 1). It is the information carried in events and their temporal separations that DEVS employs to approximate arbitrary systems. In the quantized systems approach to be discussed next events are boundary crossings and the details of the trajectories from one crossing to another are glossed over with only the time between crossings preserved.

[2] Discrete event concepts are also the basis for advanced distributed simulation environments, such as the High Level Architecture (HLA) of the Department of Defense, that employ multiple computers exchanging data and synchronization signals through message passing. [3]

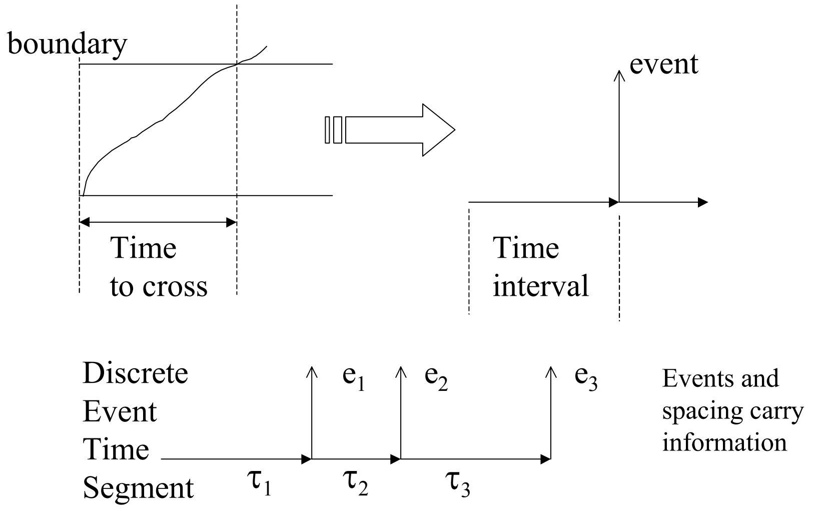


Figure 1 How continuous trajectories are abstracted into time-indexed events

Quantization

Quantization is a general process for extracting information from a continuous stream of data. When I first studied quantization, it was in the context of dynamic systems and I was able to show that any ordinary differential equation system can be approximated as closely as desired using quantization. [5]The approach, which provides an alternative to conventional numerical integration, is especially applicable in distributed simulation where components of a composite model are executing on different nodes in a network. In this context, quantization is a basic filtering technique in which continuously changing state variables, such as positions and velocities, of one component are only transmitted to other "subscriber" components over the network, when their changes exceed a threshold level called a quantum. Such quantum threshold crossings are the events and the intervals between them can be predicted so that the overall behavior can be produced by discrete event (and in particular DEVS) abstractions of the components. The larger the quantum, the fewer the state updates that are "published," but also the greater the potential deleterious effect of the message reduction on simulation accuracy. Fortunately, we have found that for many behaviors, the tradeoff of fidelity versus message reduction is very favorable – allowing available bandwidth to be utilized much more efficiently with minimal impact on fidelity. [7]

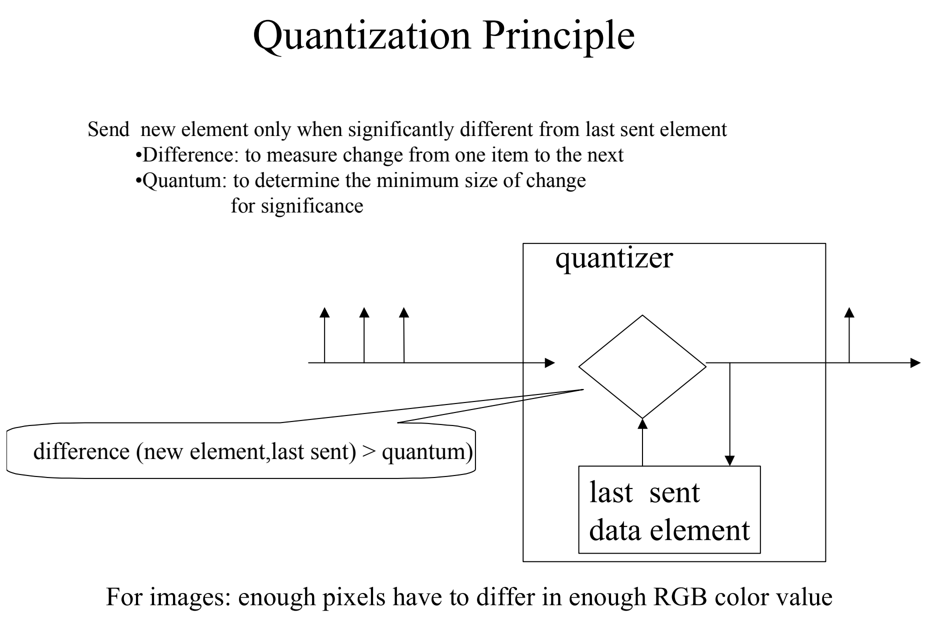

Quantization Principle
Send new element only when significantly different from last sent element
•Difference: to measure change from one item to the next
•Quantum: to determine the minimum size of change for significance
quantizer
difference (new element,last sent) > quantum)
last sent data element
For images: enough pixels have to differ in enough RGB color value

Figure 2 The generalized Quantization Principle

Quantization in Image Space

At first glance, quantization appears to be a strictly numeric process with little carryover to event abstraction in general CAS contexts. However, recently we have applied quantization concepts to extraction of events from streaming image data with surprising results. To see how this works, consider Figure 2, which depicts quantization as a general principle of information transmission from sender to receiver. At the sender's site, data elements are being continuously generated. Rather than transmit each element, a quantizer examines each one to see if it is significantly different from the one that was last sent. If so, it is transmitted and is stored as the last element sent. If not, the next element is examined, and so on.

Figure 3 illustrates a context in which quantization is applied to image capture and transmission. A web camera produces images at a fixed rate, say 30 frames/sec, which ordinarily are stored as *.jpg files in a directory. We note that JPEG is a compression technique that exploits redundancy in individual image files but does not consider redundancy in successive files. For example, if there is no motion from one frame to the next, successive files will be still be generated even though their contents are the same. Quantization, on the other hand, works on the principle that "enough" pixels have had to have differed "enough" in their (red, green, blue) color values to merit transmitting (or even storing, for some purposes). The quotations just made indicate the need for numerical values, or quanta, to make the concept operational. In experiments, we have done, typical operational values might be 15 for the color quanta (requiring at least such a difference in any of the three component values) and 10% for the pixel count quantum[3].

[3] It is interesting to note that we tried to use the compressed .jpg form directly, but even with a small motion from one frame to the next, the resulting coded version differ considerably (in length of file as well), making it hard to specify quantum levels for significant differences. RGB values are much less noisy, with a 6-sigma value of approx. 20 as indicated allowing clear separation between true change and random fluctuation.

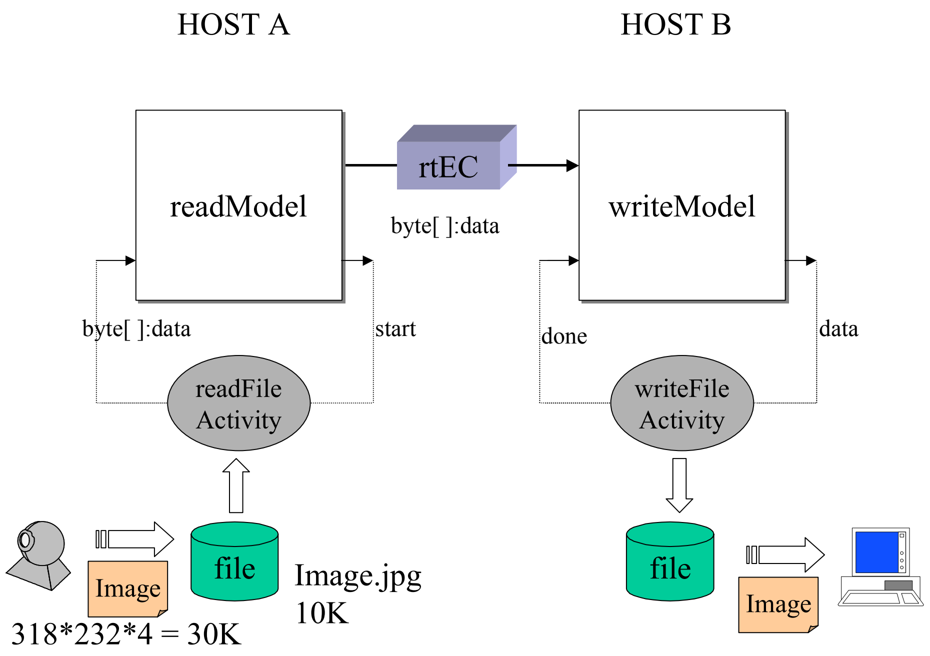


Figure 3 Transmitting images across a network

While the minimum quantum sizes are determined by noise levels, this still leaves lots of room for levels above the lower bounds. Of course, if one wants a replay of the stream as a movie that is not perceptibly different from the original, the quantum must be set at low levels. However, the interesting applications are where we can use large quanta to detect events with "semantic" content.

How can quantization, a technique with no built-in vision processing to speak of, detect events that have some meaning associated with them? Figure 4 illustrates the kind of context we have been studying. We want to detect events such as servicing a piece of equipment that are otherwise hard to implement with special devices or burdensome for a technician to record. The quantizer observers a stream of 300 frames in which a technician walks up to a machine, services it, and walks away. With appropriate settings, the quantizer extracts the five successive frames shown, as distinctly different from the base frame (in which the technician is absent) and each, from its immediate predecessor (for a reduction in frames of 5/300). We then labeled these reference frames as states in a DEVS model which also can contain expected time windows for transitions from one to the next forming an event-based control model. [8-10]If the quantizer detects a significant change in the image stream within the time window allowed by the current state of the model, then a transition is triggered to the next state[4] To record a complete service occurrence, the model must be traversed from the base state back to itself through the given cycle, with all time windows honored. Otherwise, an alarm can be triggered to indicate the occurrence of an abnormality, such as the technician entering and leaving before actually touching the equipment. Implementation of this image quantization and interpretation process was done on the distributed DEVS/CORBA real-time execution and simulation environment [11, 12]which allows writing DEVS models as if they were to be simulated, and after testing in simulation, having them executed as agents operating in real time.

To summarize, quantization appears to be a fundamental process by which a temporal succession of events is extracted from a continuous stream of data. It is based on detecting significant change between each datum and the last selected reference item. Time intervals between such events (change in reference datum)

[4] To trigger this transition at a remote location, only the event occurrence not even the frame has to be sent, thus further reducing bandwidth requirements. However, by sending the frame, checking against state specific templates can be done to assure that the right state is maintained in the model.

carry information and can be used for example to trigger transitions in a model whose states correspond with the events.

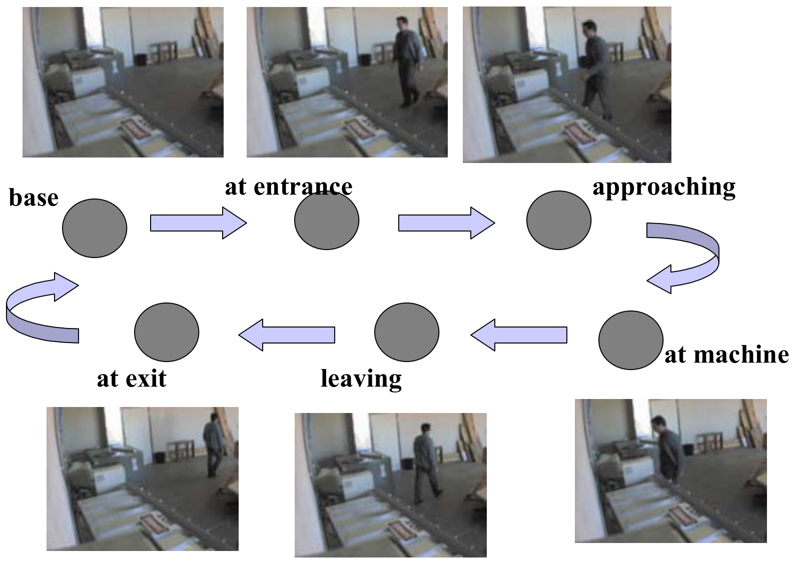


Figure 4 Video observation of equipment servicing

Fast Discrete Event Neuron Architectures

Research started in my lab as early as 1995 to employ DEVS abstractions to capture the many features of biological neurons that were not represented in conventional artificial neural networks and to exploit these capabilities to perform intelligent control tasks. [13, 14] In the meantime, other work employing event-driven simulation [15]and on alternative neuron architectures has yielded a strong argument for "one spike per neuron" processing in natural neural systems. [16] "One-spike-per-neuron" refers to information transmission from neuron to neuron by single pulses (spikes) rather than pulse trains or firing frequencies. A face recognition multi-layered neural architecture based on the one-spike, discrete event principles has been demonstrated to better conform to the known time response constraints of human processing and also to execute computationally much faster than a comparable conventional artificial neural net[5]. [17, 18] The distinguishing feature of the one-spike neural architecture is that it relies on a temporal, rather than firing rate, code for propagating information through neural processing layers. This means that an interneuron fires as soon as it has accumulated sufficient "evidence" and therefore the latency to the first spike codes the strength of this input. Single spike information pulses are thus able to traverse a multi-layered hierarchy asynchronously and as fast as the evidential support allows. Thorpe's research team [16, 17]has also shown that "act-as-soon-as-evidence-permits" behavior can be implemented by "order-of-arrival" neurons which have plausible real world implementations. Such processing is invariant with respect to input intensity because latencies are uniformly affected by such changes. Moreover, coding which exploits firing order of neurons is much more efficient than a firing-rate code which is based on neuron counts.

[5] The face recognition layered net was executed by a discrete event simulator and took between 1 and 6 seconds to recognize a face on a pentium PC vs. several minutes for a conventional net on a SGI Indigo. Recognition performance in both cases was very high. The authors employed a training procedure which, while effective, is not plausible as an in-situ learning mechanism.

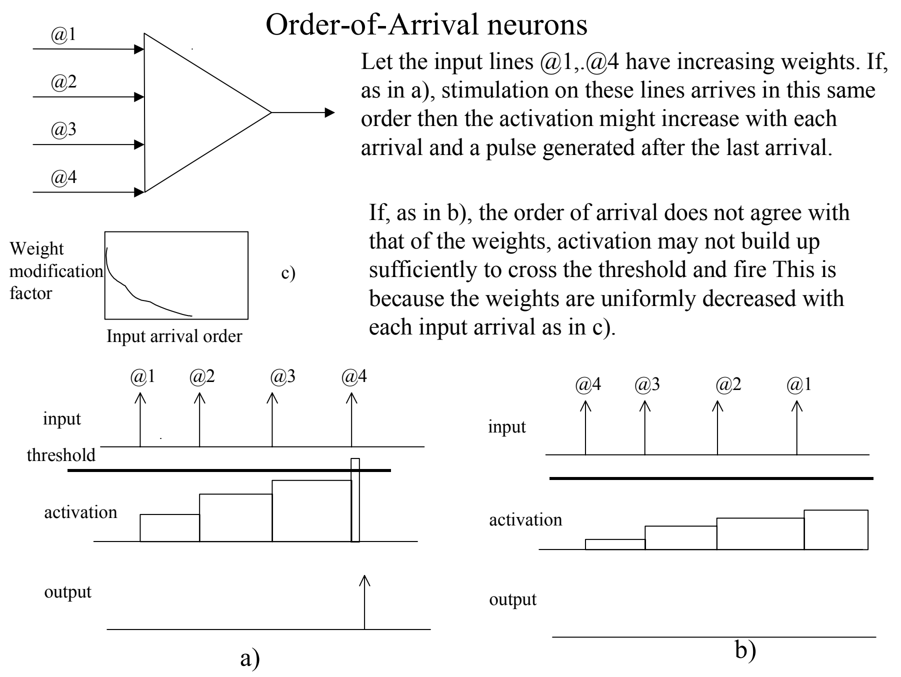
The discrete event nature of the one-spike concept suggests that it might be an ideal candidate as the basic building block in an “end-to-end” processing system for small, fast reactive “nervous systems”. Therefore we have formulated the discrete event abstractions underlying the one-spike-per-neuron concept, and expressed them in DEVS. It is remarkable, indeed, how well the one-spike-per-neuron conceptual framework fits the definition of discrete event abstraction provided earlier. The events here are threshold crossings which generate discrete spikes while inter-event temporal separations include the latencies between input and output spikes.

Strength-to-Latency Coding

The basic concept that supports discrete event abstaction of neural behavior is strength-to-latency coding. Here the strength of the input of an evidence gathering neuron (such as sensory neuron) is coded in the latency of its output response for downstream neurons. In other words, the greater the stimulation of an input volley (evidence) the quicker the generation of a corresponding output spike. Thus a neuron with lots of evidentiary support will be “heard” earlier by neurons in the next processing layer than one with low or no input strength. Dispersion in such latencies sets the stage for neurons that are sensitive to the order of arrival of spikes.

Order-of-Arrival Neurons

The behavior of an order-of-arrival neuron is illustrated in Figure 5. An input train arrives on the input lines in the order of their weights accumulates maximum activation and may cause the neuron to fire if this exceeds the threshold. Any other order of arrival will accumulate less activation and therefore, depending on the threshold level, may not generate an output spike. Thus the neuron can discriminate among different order-of-arrivals of stimuli. This ability to distinguish between N! input patterns (where N is the number of input wires) thus supports a combinatorially more efficient information code than one based on the number of stimulated input wires rather than their order of stimulation [16]


Order-of-Arrival neurons

Let the input lines @1,.@4 have increasing weights. If, as in a), stimulation on these lines arrives in this same order then the activation might increase with each arrival and a pulse generated after the last arrival.

If, as in b), the order of arrival does not agree with that of the weights, activation may not build up sufficiently to cross the threshold and fire This is because the weights are uniformly decreased with each input arrival as in c).

Figure 5 Order-of-arrival Neurons

End-to-End Processing Layers

Consider the hypothesis that the “act-as-soon-as-evidence-permits” principle will make it practical to implement a fast processing layered architecture, including all its sensory, cognitive, actuator and communication related components, within real time processing, memory and energy constraints. To study this hypothesis, we developed the “end-to-end” layered architecture for a reactive nervous system shown in Figure 6 along with the kinds of neurons that are found in each layer. After describing their basic processing roles, we will discuss the discrete event representations of the neurons.

- Sensory layer neurons react directly to incoming energy (in various forms such as visual or infrared electronmagnetic waves, sonar, etc.) These neurons perform the analog-to-latency coding just discussed.

- Fusion/Analysis neurons fuse the data collected from the various sensors into some stereotyped situations that can be further related to reactive courses of action. These neurons operate on the order-of-arrival principles discussed above.

- Priming of alternative candidates for behavioral course of action is also done by order-of-arrival neurons.

- Decision, i.e., selection from the candidates, is performed the by winner-take-all neurons.

- Action sequencing plays out the memorized sequence of actions associated with a selected course of action and is done by event-based control neurons.

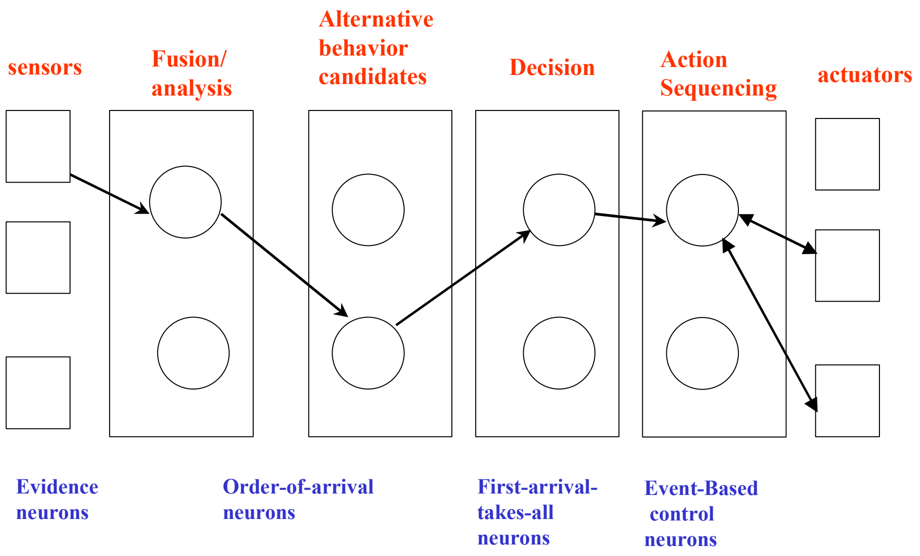


Figure 6 An "end-to-end" layered architecture to establish neuron behavior requirements

Synchronizing Between Layers

One problem that emerges immediately is that using strength-to-latency coding, operation is fundamentally asynchronous. Unlike in conventional models such as cellular automata, there is no global clock tick to synchronize states of neurons at a recurring time step. Consequently, as illustrated in Figure 7, laggard spikes from earlier percepts may interfere with spikes stimulated by later percepts. One solution is to place a time-out on the response of a neuron -- it is reset to its initial state after a given duration.

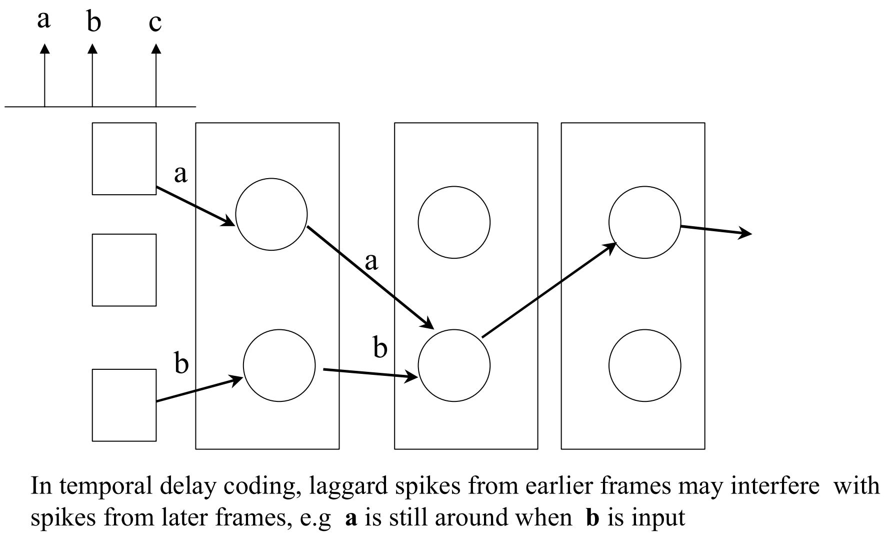

a
b
c
a
a
b
b
In temporal delay coding, laggard spikes from earlier frames may interfere with spikes from later frames, e.g **a** is still around when **b** is input

Figure 7 Laggard pulses in stength-to-latency information transmission

DEVS Neuron Models

Having considered the roles that various neurons will have to play in the just outlined prototype architecture, we can develop requirements for their basic behavioral properties. Note these requirements, while inspired by the biological origins of discrete event neural abstractions, were found to be logically required in implementing the above prototype[6]. Our DEVS neurons

- will have the ability to respond to order of arrival of external events on their input ports
- will be controled by passage of time, such as time windows and time outs
- can delay firing to enable competition in sending output to next stage
- must be synchronizable through an external reset event.

Figure 8 presents a graphic representation of a generic DEVS model that satisfies these requirements.[7] The model has four main phases (control states): *receptive, refract, active* and *fire*, each with an associated time

[6] With the implication that they offer explanations for why the biological properties assume the form they do.
[7] For a more complete explanation of the graphical notation used and its correspondence to the formal representation and computer implementation of DEVS models see. [5].

duration. The actual durations are parameters of the model which range from 0 to infinity. Their assignments produce different specialized behaviors that are derived from the generic model.

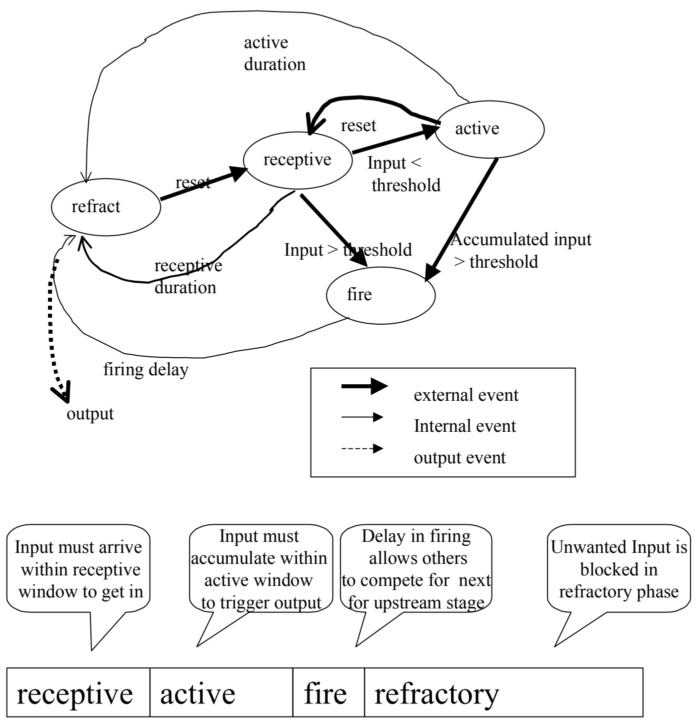

active duration
reset
active
receptive
Input < threshold
reset
refract
Input > threshold
Accumulated input > threshold
receptive duration
fire
firing delay
output
external event
Internal event
output event
Input must arrive within receptive window to get in
Input must accumulate within active window to trigger output
Delay in firing allows others to compete for next for upstream stage
Unwanted Input is blocked in refractory phase
receptive
active
fire
refractory

Figure 8 DEVS model of neurons satisfying the behavioral requirements

The model typically starts in the receptive phase. If an input arrives during the receptive period that is less than the threshold, then the neuron enters the active phase, where it waits for further inputs. If accumulated input exceeds the threshold before the active period has expired, then fire phase is entered. Also, if an above-threshold input arrives during receptive period, fire phase is entered directly. After a delay, an event (representing a pulse or spike) is produced on the output port. After firing, the model enters the refractory phase, where it is unresponsive to inputs. The active phase also times out to the refractory phase (if above threshold input is not accumulated). The reset input, occurring during the refractory period, puts the model back to the receptive phase.

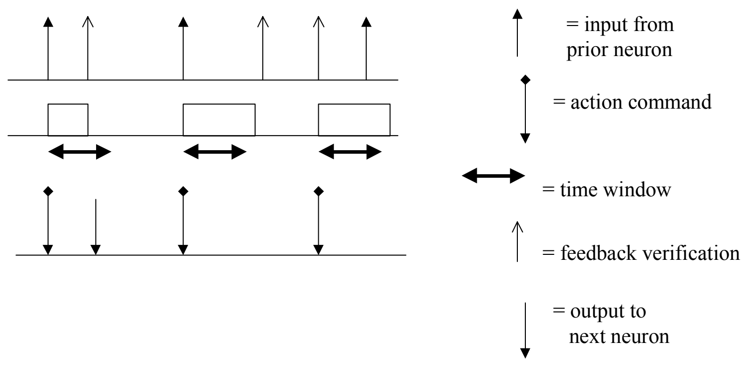
The generic model can be specialized to realize the behaviors of the following:

- Evidence Neurons (at the sensory layer) – Physiologically, these have been identified as "integrate-and-fire" neurons [16]and represented as leaky integrators with threshold-based firing. With constant inputs arriving periodically, an output will be generated once the threshold has been reached. The output period is inversely related to the strength of the input thus implementing the analog-to-delay coding discussed earlier.

- Order-or-Arrival Neurons – these are implemented with appropriate weight settings as discussed earlier.

- First-Arrival-Takes-All Neurons – these neurons implement winner-take-all behavior based on first arrival. Metaphorically, the neuron with the first spike to arrive from the previous processing stage, closes the door for pass through of later arrivals. This approach works much faster than conventional winner-take-all circuits. [16]Using the generic DEVS neuron, the lockout behavior can be accomplished by establishing mutual cross-inhibition (negative weights for inputs from competing predecessors). We'll discuss a related example in the context of fast and frugal heuristics in a moment.

- Event-based Control Neurons – these implement event-based control concepts discussed earlier in the context of image quantization. Neurons are connected in a series to control a sequence of discrete actuations forming a composite action. In event-based control, verification feedback from the current actuation (as in proprioceptive feedback) is required to fall within a time window before the next action can be activated, as illustrated in Figure 9. The realization by the generic DEVS neurons employs the time-out associated with the active phase.



If a neuron receives input from predecessor it emits action command and waits for feedback verification If feedback from actuator is within the allowed time window, the action sequence continues to the successor.

Figure 9 DEVS neuron implementation of event based control for output effectors

DEVS Neurons: Time, Space, and Energy Constraints

We pause to reconsider Pinker's characterization of the space, time and resource imperatives for real world information processing in the context of our DEVS neuron models. In terms of space frugality, as indicated earlier, order-of-arrival coding is much more efficient in its use of neurons than is rate-based coding. Moreover, the "act-as-soon-as-evidence-permits" principle is clearly compatible with the demands of quick reaction under time pressure. Further the time-outs associated with phases can be linked to limitations on the energy consumption necessary to maintain active states. Likewise the refractory phase can be associated with minimal energy consumption (or restoration of energy in the biological case). Of course, the underlying DEVS abstraction, concentrating on events and their timing, is efficient in both processing and communication as discussed earlier.

Fast and Frugal Heuristics

Single spike neurons and "act-as-soon-evidence-permits" neural architectures offer a discrete event computational infrastructure for CAS behaviors constrained by time, space and energy resources. But what is lacking in such discrete event neural architectures is an overall cognitive system framework that characterizes the behaviors to implement. The "fast frugal and accurate" (FFA) perspective on real word intelligence [19, 20]provides such a framework. This perspective recognizes that the real world is a threatening environment where knowledge is limited, computational resources are bounded, and there is no time for sophisticated reasoning. Unfortunately, traditional models in cognitive science, economics, and animal behavior have used theoretical frameworks that endow rational agents with full information of their environments, unlimited powers of reasoning and endless time to make decisions. Evidence and theory from disparate sources have been accumulating that offer alternatives to the traditional paradigm. Indeed, simple building blocks that control attention to informative cues, terminate search processing, and make final decisions can be put together to form classes of heuristics that have been shown in many studies to perform at least as well as more complex information-hungry algorithms. Moreover, such FFA heuristics are more robust than others when generalizing to new data since they require fewer parameters to identify. They are accurate because they exploit the way that information is structured in the particular environments in which they operate. It is important to note that FFAs are a different breed of heuristics. They are not optimization algorithms that have been modified to run under computational resource constraints, e.g., tree searches that are cut short when time or memory run out. Typical FFA schemes exploit minimal knowledge such as object recognition and other one-reason bases for choice making under time pressure, elimination models for categorization, and satisficing heuristics for sequential search. An organism's FFAs are essentially *models* of the real environment in which it has found its niche and to which it has (been) adapted.

In his radical departure from conventional rational agent formulations, Simon asserted the "bounded rationality" hypothesis. This states that an agent's behavior is shaped by the structure of its task environment and it's underlying computational abilities. [21]Fast and frugal heuristics are mechanisms that a mind can execute under limited time and knowledge availability and that could have possible arisen through evolution. As an illustration of an FFA we briefly discuss an example, the "take the best" inferencing heuristic which employs only a fraction of the available knowledge and stops immediately when the first answer is found (rather than the best answer). "Take the best" does not attempt to integrate all available information into its decision, it is non-compensatory, non-linear, and can violate transitivity, the cannon of rational choice.

The table illustrates the knowledge framework in which "take the best" operates. Consider a table having in the order of one-hundred cities and in the order of 10 attributes of each, with each cell containing a + (city has the attribute), - (city does not have the attribute) and ? (don't know). Given a pair of cities, the heuristic infers an answer to the question of which is larger, by examining attributes as predictors, accepting the first for which there is exactly one + distinguishing the pair . To explain: a city that has the attribute is presumed to be larger than one that doesn't or with which one is unfamiliar; an attribute provides no basis for discrimination, if both cities have it. The order of examination is be based on *predictive validity*, which is defined in terms of the correlation between the attribute and the criterion. More specifically, the predictive validity of an attribute is the relative frequency with which it predicts the criterion (see [19]for a detailed discussion). For example, the predictive validity of HasFootballTeam is high since cities with football teams tend to be larger than those without them, the predictive validity of IsOnCoast is lower since many, but not all cities on the coasts of the US; are larger than interior cities while IsStateCapitol is low since very few state capitols are large (in fact Phoenix is a major exception). Gigerenzer and Goldstein [20] show that this model works well for explaining actual subject data and though simulation, that it performs at about the same accuracy levels as conventional algorithms that sum and weight the attribute information, while being much faster. Interestingly, the heuristic can provide answers even when most of its knowledge is missing (most cells are ?). It also has interesting properties such as performing best in a state of less than full knowledge (illustrating a "less is more" principle: familiarity, and the lack theieof, provide bases for decision making in a real world of attention-grabbing features. [19]

City/ property	**Is StateCapitol**	**HasFootballTeam**	**IsOnCoast**

New York	-	+	+
Phoenix	+	+	?
LA	?	?	+
…			

If fast and frugal heuristics characterize human bounded rationality, then they ought to be implementable by discrete event neural architecture discussed earlier. Indeed, let us sketch a construction to show this to be the case. Figure 10 depicts a network of DEVS neurons that implements "take the best". Neurons representing cities connect to subnetworks representing attributes in a manner reflecting the available knowledge. We will discuss these in more detail below. Other than this modification, the implementation employs the same primitives discussed earlier, including the importance coding and first-to-arrive-takes all properties.

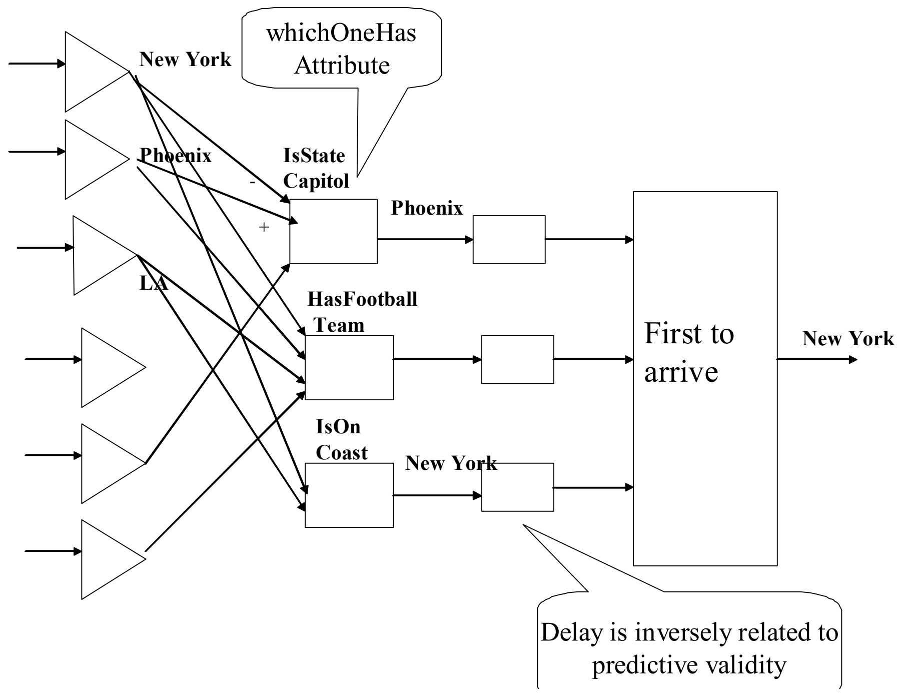
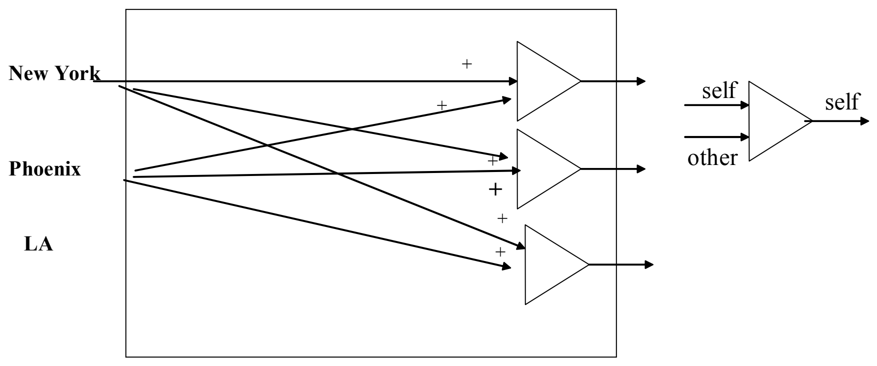


Figure 10 DEVS neural architecture implementing "take the best" inferencing

In Figure 10, the first layer of neurons generate pulses when the cities they represent are mentioned. So the question, “Which is larger, New York or Phoenix?” causes pulses to emerge from neurons New York and Phoenix.. The second layer components represent attributes in the knowledge table above. The connections from the first layer to the second represent the associations in a direct manner. For example, there is a positive connection from Phoenix to IsStateCapitol but none to IsOnCoast, representing lack of knowledge (? in the table). On the other hand, there is a negative connection from New York to IsStateCapitol corresponding to the known fact in the table.

Figure 11 shows an attribute subnetwork which is a coupled model of DEVS neurons. Such an attribute component has the *cancellation* property: -- an output emerges if a single positive input is received, but the firing is slightly delayed so it can be cancelled by a second positive input. To illustrate this property of DEVS neurons, since both New York and Phoenix have football teams, each fires a corresponding neuron. However, at the same or slightly later each pulse also cancels the other's output pulse. (We note that simultaneity is not required, only arrival of a second pulse within the firing delay.) Thus, there is no output from the HasFootballTeam attribute. On the other hand, Phoenix will be the output from IsStateCapitol, since the pulse generated in the Phoenix neuron is unaffected by the negative connection from New York.

Which one has the Attribute, e.g. HasFootballTeam



Components are simple neurons with cancellation.
If only one + arrives it on self port, it emerges after a firing time.
Otherwise, there is also a + input on the other port which cancels
the firing to annihilate the output

Figure 11 An Attribute Subnetwork

Now consider the response to pulses from Phoenix and New York generated in the first layer by the question asking for a relative size judgment between Phoenix and New York. When the pair of pulses so generated arrives at the attribute components, the winners of IsStateCapitol, HasFootballTeam and IsOnCoast will be Phoenix, none, and New York, respectively. The outputs of the second layer continue on to the third layer which implements winner-take-all behavior based on the "first-to-arrive-wins" principle. More specifically, pulses representing the winners of the attribute subnetworks are delayed by an amount that relates inversely to the respective *predictive validities* of the attributes. Thus, in the race between Phoenix and New York pulses, emerging from the attribute layer, New York arrives first, since the delay of the "IsOnCoast" attribute is smaller than that of the "IsStateCapitol" attribute. As earlier suggested, this behavior is characteristic of the discrete event neuron architecture. Distinct from conventional neural nets (where strength of association is assume to be manifest in higher firing counts), in "act-as-soon-as-evidence-permits" networks, the strength of association shows up as faster travel of pulses from one processing layer to the next. Those that arrive at the last stage decision point govern the final output.

The above parallel/distributed reformulation of "take-the-best" offers a parallel alternative to the sequential implementation of fast and frugal heuristics (Gigerenzer and Todd 1999). In contrast to sequential examination of attributes they are all stimulated in parallel. "Take the best" is fast and frugal because it typically examines only a few attributes before finding one that makes the needed discrimination[8]. In our implementation, all discriminating attributes are found concurrently then subject to a winner-take-all competition based on predictive validity. Thus the parallel implementation can be faster (depending on the predictive validity delay) with the same accuracy. However, since it examines all attributes it is not frugal. Perhaps frugality can be introduced by introducing firing delays between the first and second layers inversely related to the discrimination power of attributes – those that are highly discriminating will get the pulses earlier than those that re less so. We leave this approach, which tends to balance discriminating power and predictive validity, for future examination.

Dynamic Binding and Temporal Relations

There has been increasing interest in how time enters into neural processing in brains. With recent experimental work summarized by Singer, [22]one major question being asked is whether synchronization among cell firings plays a major role in fusing the results of lower level processing into higher level elements for further processing. The attractive feature of such dynamic binding is that it can coordinate widely dispersed cells without requiring a combinatorial explosion of neurons for explicitly representing all possible conjunctions of input elements. The discrete event neural network presented above is not based on dynamic binding through synchronization. Indeed, creating simulation models which work on dynamic binding principles seems quite challenging. Paradoxically, finding synchrony in unstructured networks seems not very hard. Indeed, Amit [23]offers some insight into the difficulty involved in inferring the information-bearing role of neural firing synchronization in large networks, both in simulations or real brains. He shows that cross-correlations of paired neuron spike trains can be obtained in simulations of randomly connected integrate-and-fire neurons that are qualitatively similar to those obtained from behaving monkeys. Moreover, theory suggests these synchronies have little to do with upstream processing. [24]

Efficiency of Discrete Event Simulation

For reasons of modeling efficacy and simulation feasibility, we suggest that discrete event modeling provides the right level of abstraction to address the challenges just described. Mattia and colleagues [Mattia, 2000 #101 employ an event-driven approach for large-scale simulations of recurrent neural networks of spiking neurons and plastic synapses. The approach exploits the fact that the dynamic variables of both neurons and synapses evolve deterministically between any two successive spikes -- thus tracking the arrival of spikes enables unequivocal prediction of neural dynamics. Compared with conventional synchronous simulation. [25] the result is a drastic reduction of the computational load per neuron which grows linearly with the number of neurons and is only about 6% more with variable synapses.

It is enlightening to place Mattia's simulation complexity comparison in the context of the more fundamental discrete event conceptual framework presented here. In discussion surrounding Figure 1 we indicated that discrete event abstraction can retain essential state and timing information while abstracting away details in underlying continuous trajectories. This said, there ought to be a fundamental impact on system simulation that exploits such abstraction and there is when compared with the conventional discrete time approach − which unfortunately, is still the most common by far, in natural sciences. We can indicate this is in a quantitative manner by examining a DEVS coupled model as in Figure 12. Let there be N components, each sending outputs to an average of C receivers. Such outputs, generated at internal state transitions, are assumed to be described by a Poisson process with rate, ν_{int}. Under random connectivity conditions, every component receives inputs at the rate, $\nu_{ext.} = CN\ \nu_{int}$. Now, a DEVS simulator only computes updates at internal and external events. At each component these occur at the combined rate $\nu = \nu_{int} + \nu_{ext.} = (1+CN)\ \nu_{int.,}$ a linear dependence on N (again assuming validity of the Poisson description).

[8] Highly predictive attributes tend also to be less discriminating than lower predictive ones, thus accounting for number of steps before decision (see Gigerenzer and Todd 1999).

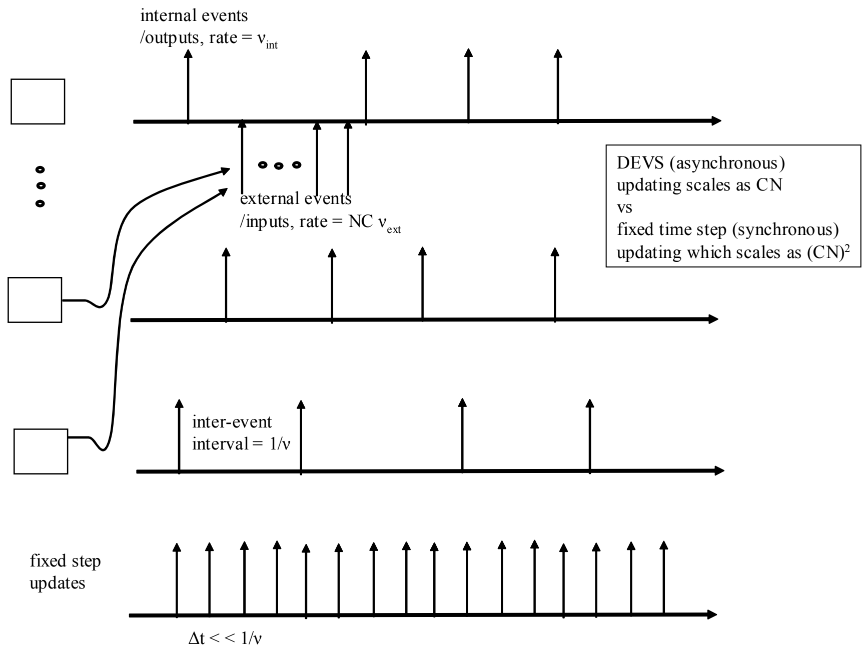


Figure 12 DEVS Network to illustrate efficiency of discrete event simulation

In comparison, a fixed time step simulation (or synchronous approach as it is commonly called) is must work at the rate of $(1+CN)/\Delta t$ updates per second per component, where Δt is the time step. Now, the key difference is that Δt must be small enough to keep the probability of missing an event between updates very small. Indeed, Δt must be much less than the average interval between events is $1/\nu = 1/(1+CN)\,\nu_{int}$. Thus, the update rate per component is greater than $(1+CN)^2\,\nu_{int}$. This quadratic dependence on N is clearly not scalable to large numbers of components, N. Stated another way, the ratio of fixed time step updates to discrete event updates is (1+CN), which grows increasingly worse for large C and/or large N. A similar conclusion is presented for differential equation simulation in, [5] reinforcing the advantages of discrete event modeling and simulation for both continuous and discrete systems (or better systems in general, no matter what their dynamics appear to be characterized by).

The efficacy and efficiency of discrete event abstraction is illustrated in a recent DEVS formulation of Epstein and Axtell's Sugarscape [26] (available from www.acims.arizona.edu), which explored the application of agent-based simulation to social science.

Computation with Pulsed Neural Networks: Shortest Path Solvers

Another direction pointing toward discrete event neurons is suggested by the recent interest in the computational capabilities of "pulsed neural networks". [27]Interestingly, research reported in this direction, in the main, does not take the full step toward discrete event abstraction and hence toward DEVS representation. Such abstraction offers a simplification of concept that then stimulates further advances in the new space opened up by the simplified conceptual framework. For example, adoption of switching (boolean, combinatorial) logic allowed design of digital computers and, in freeing the abstraction from its technological embodiment, gave birth to a completely new field of computer science. In the case of switching logic, the technological embodiment had to be in place to ensure the stability the two distinct states representing the abstractions 0 an 1 and the transitions specified by design. In the case of discrete event abstraction of pulsed neurons, the DEVS formalism and the DEVS simulation/execution environments provide the support needed to work with the abstraction as a reality. The reader is invited to compare the simplicity of the DEVS representation with the representations found in [27]and note the absence of next-level "end-to-end" architectural considerations in the latter.

Although space precludes a detailed discussion here, it should not come as a surprise that very simplified versions of the generic DEVS neurons suffice to provide shortest path solutions to directed graphs. Indeed, the shortest path in such a graph, whose arcs represent time durations, can be regarded as an abstraction of the "act-as-as-soon-as-evidence-permits" processing in multilayer nets. In contrast, finding long paths cannot be done with the DEVS neuron as described[9]. This suggests that an evolutionary explanation for the structure and behavior of biological neurons if indeed they operate on discrete event principles -- they are optimized for operation in a world requiring synthesis of quick multi-step reactions as dictated by shortest paths in graphs of dependent actions.

Conclusions

We have shown how discrete event abstraction captures information in rich continuous data streams in terms of events and their timed occurrences. Quantization is a general process that can perform such abstraction. Evidence from recently developed models of neurons and neural processing architectures and from fast and frugal heuristics provide further support for the centrality of discrete event abstraction in modeling complex adaptive behavior when the constraints of space, energy and time are taken into account. More speculatively, if the conjecture on the central role of dynamic binding in information processing holds true, the different modes of temporal processing   may prove critical to understanding how the human mind works. More generally, any CAS must operate within the constraints imposed by space, time, and resources on its information processing. In particular, working in real time, and with limited bandwidth, a distributed CAS must evolve its computation and communication channels to route the right information to the right consumers at the right times. I suggest that the discrete event paradigm provides the right level of abstraction to tackle these issues.

[9] For example, rather than firing on the first received pulse, and then refracting, finding a longest path requires passing on every received pulse while waiting for an even later straggler.

References

1. Holland, J.H., *Complex Adaptive Systems.* Daedalus, 1992. **121**(1): p. 17-30.
2. Pinker, S., *How the mind works*. 1997, New York, NY: W.W. Norton.
3. Defense, D.o., *High Level Architecture Interface Specification, Version 1.0*, . 1996, Defense Modeling and Simulation Organization,available via http://msis.dmso.mil.
4. Zeigler, B.P., *Theory of Modelling and Simulation*. 1976, New York: John Wiley (Under Revision for 2nd Edition 1998).
5. Zeigler, B.P., T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*. 2 ed. 2000, New York, NY: Academic Press. 510.
6. Zeigler, B.P. and H. Sarjoughian. *Support for Hierarchical Modular Component-based Model Construction in DEVS/HLA*. in *SIW*. 1999. Orlando, FL.
7. Zeigler, B.P., *et al.* *Bandwidth Utilization/Fidelity Tradeoffs in Predictive Filtering*. in *SIW*. 1999. Orlando, FL.
8. Zeigler, B.P. *DEVS Representation of Dynamical Systems: Event-based Intelligent Control*. in *Proceeding of IEEE*. 1989.
9. Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. 1990, San Diego,CA: Academic Press.
10. Luh, C. and B.P. Zeigler, *Abstracting Event-Based Control Models for High Autonomy Systems.* IEEE Trans. System, Man and Cybernetics, 1993. **23**(1): p. 42-54.
11. Cho, Y.K., *et al. Design Considerations for Distributed Real-Time DEVS.* in *AIS 2000*. 2000. Tucson, AZ.
12. Harrison, T.H., *et al.*, *The Design and Performance of a Real-time CORBA Event Service.* IEEE J. on Selected Areas in Communications, 1999.
13. Vahie, S. and N. Jouppi. *Dynamic Neuronal Ensembles: A New Paradigm for Learning & Control*. in *AI, Simulation and Planning in High Autonomy Systems*. 1996. San Diego: Web: www-ais.arizona.edu.
14. Vahie, S., *Dynamic Neuronal Ensembles: Neurobiologically Inspired Discrete Event Neural Networks*, in *Discrete Event Modeling and Simulation Technologies*, H. Sarjoughian, Editor. 2000, Springer-Verlag.
15. Watts, L., *Event-driven simulation of networks of spiking neurons*, in *Advances in neural information processing systems*. 1994, Morgan Kaufmann: San Mateo, CA. p. 927-934.
16. Gautrais, J. and S. Thorpe, *Rate Coding Versus Temporal Coding: A Theoretical Approach.* BioSystems, 1998. **48**(1-3): p. 57-65.
17. Rullen, R.V., *et al.*, *Face Processing Using One Spike Per Neurone.* BioSystems, 1998. **48**(1-3): p. 229-239.
18. Delorme, A., *et al.*, *SpikeNET: A simulator for modeling large networks of integrate and fire neurons*, in *Computational neuronscience: Trends in Research*. 1999, Elsevier Science: Amsterdam. p. 989-996.
19. Gigerenzer, G. and P.M. Todd, *Simple Heuristics That Make Us Smart*. 1999: Oxford University Press.
20. Gigerenzer, G. and D.G. Goldstein, *Reasoning the Fast and Frugal Way: Models of Bounded Rationality*, in *Judgment and Decision Making*, T. Connolly, Editor. 2000, Cambridge U. Press. p. pp. 621=650.
21. Simon, H.A. and A. Newell, *Information Processing in Computer and Man.* American Scientist, 1964. **52**: p. 281-300.
22. Singer, W., *Neural Synchrony: A Versatile Code for the Definition of Relations?* Neuron, 1999. **23**: p. 49-65.
23. Amit, D.J., *Simulation in neurobiology - theory or experiment?* TINS, 2001. **in press**.
24. Amit, D.J., *Is synchronization necessary and is it sufficient?* http://www.fiz.huji.ac.il/staff/acc/faculty/damita/papers.html, 2000.
25. Bower, J.M. and D. Beeman, *The book of GENESIS*. 1998, New York: Springer-Verlag.

26. Zaft, G.C. and B.P. Zeigler. *Discrete Event Simulation and Social Science: The XeriScape Artificial Societ***y**. in *SCI 2002*. 2002. Orlando, FL.
27. Maas, W. and C.M. Bishop, *Pulsed Neural Networks*. 1999, Cambridge, MA: MIT PRess. 377.