information

Article

DEVS-Based Building Blocks and Architectural Patterns for Intelligent Hybrid Cyberphysical System Design

Bernard Zeigler [1,2]

[1] Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721, USA; zeigler@rtsync.com
[2] RTSync Corp., Chandler, AZ 85226, USA

Abstract: The DEVS formalism has been recognized to support generic open architectures that allow incorporating multiple engineering domains within integrated simulation models. What is missing for accelerated adoption of DEVS-based methodology for intelligent cyberphysical system design is a set of building blocks and architectural patterns that can be replicated and reused in system development. As a start in this direction, this paper offers a notional architecture for intelligent hybrid cyberphysical system design and proceeds to focus on the decision layer to consider DEVS models for basic behaviors such as choice of alternatives, perception of temporal event relations, and recognition and generation of finite state languages cast into DEVS time segments. We proceed to describe a methodology to define DEVS-based building blocks and architectural patterns for design of systems employing fast, frugal, and accurate heuristics. We identify some elements of this kind and establish their status as minimal realizations of their defined behaviors. As minimal realizations such designs must ipso facto underlie any implementation of the same cognitive behaviors. We discuss architectures drawn from the cognitive science literature to show that the fundamental elements drawn from the fast, frugal, and accurate paradigm provide insights into intelligent hybrid cyberphysical system design. We close with open questions and research needed to confirm the proposed concepts.

Keywords: modeling & simulation; DEVS; building blocks; architectural patterns; intelligent systems; hybrid systems; cyberphysical systems; system design; neural networks; brain models

Citation: Zeigler, B. DEVS-Based Building Blocks and Architectural Patterns for Intelligent Hybrid Cyberphysical System Design. *Information* **2021**, *12*, 531. https://doi.org/10.3390/info12120531

Academic Editor: Willy Susilo

Received: 30 November 2021
Accepted: 14 December 2021
Published: 20 December 2021

1. Introduction

Discrete Event System Specification (DEVS) and its extensions to hybrid modeling and simulation [1–3] are increasingly being adopted as the preferred approach to intelligent hybrid (continuous and discrete) cyberphysical system design [4–9]. After decades of developments in its theory, software support, and breadth of applications, the DEVS formalism has been recognized to support generic open architectures that allows incorporating multiple engineering domains within integrated simulation models. DEVS enables formal and complete description of hybrid model components and subsystems. DEVS-based software tool sets provide atomic model and hierarchical coupled model specifications that support graphical description of the internals and interfaces of component behavior combining energy, material, and information flows. The hybrid DEV&DESS formalisms enables expressing differential and algebraic equations for energy-related internal variables intermixed with discrete behavior described in state-based system form. Finally, transparent implementation of the canonical DEVS abstract simulator for handling events and equations enable design of dedicated simulation functionality.

What is missing for the accelerated adoption of DEVS-based methodology for intelligent cyberphysical system design is a set of building blocks and architectural patterns that can be replicated and reused in system development. In this paper, we first review the DEVS formalism in relation to hybrid cyberphysical systems and offer a functional architecture for these systems. Taking this architecture as representative of state-of-the-art, we

review the architectural features presented in [10,11] in order to examine them as reusable building blocks and architectural patterns. We then present an approach to establishing that a proposed model is a minimal realization of a given cognitive behavior. We also show how to create DEVS models at the state description level that generate the defined behaviors. The state level description sets up the design of a building blocks that can be employed in network applications to establish the generic reusability of the minimal realization. We show how the network form is typically closer to potential plausible biological and other implementations. The established minimality of realization allows us to claim that the latter realizations must be a homomorphic image of any such realization. We close with an architecture drawn from the cognitive science literature where parallel systems compete to control the response to environmental conditions and show that the fundamental elements drawn from the fast, frugal, and accurate heuristics paradigm [12–14] provide insights into this form of intelligent hybrid cyberphysical system design. We close with open questions and research needed to confirm the proposed concepts.

1.1. Review of DEVS Abstractions for Brain Architectures

DEVS is a system theoretic characterization of discrete event simulations based on abstraction of events and time intervals from continuous data streams [15–17]. Such abstractions carry information that can be efficiently employed, not only in simulation, but also in accounting for the real-world constraints that shape cognitive information processes [18,19]. An abstraction 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 [1]. Discrete event models of neurons, neural processing architectures, and "fast frugal" bounded rational decision-making models [12] offer the abstractions for capturing cyber-physical cognitive systems structure and behavior. While other formalisms allow representation of space and related resources, only discrete event models offer the additional ability to explicitly and flexibly express time and its essential constraints on system behavior and structure. 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 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.

Definitions for state-based realization of cognitive behaviors based on mathematical system theory and DEVS fundamentally include temporal and probabilistic characteristics of neuron system inputs, state, and outputs [1] and provide a solid system-theoretical foundation and simulation modeling framework for the high-performance computational support of such applications. Spiking neural nets (SNN), [20–25] a form of hybrid continuous discrete event abstraction, have demonstrated potential for solving complicated time-dependent pattern recognition problems because of their inclusion of temporal and dynamic behavior [3]. Realizations of SNN's in DEVS have been shown [25,26]. 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. Exploiting asynchronous behavior and sparsity of events, activity, and fan-out that seems to characterize real brain processing, neural nets of large sizes (millions of neurons and billions of synapses) can be simulated on a desktop computer [24]. For the same reasons, DEVS can support fast and energy efficient hardware implementation [25] of spiking neural nets (SNN) and other such models. In particular, DEVS supports the abstractions underlying neural net models for which:

(1) The neural elements (neurons, synapses, others to be discussed) have discrete states, DEVS provides an intuitive and expressive state-based modeling formalism.

(2) The neural elements send and receive discrete signals, which are effectively modeled by message exchange in the DEVS formalism.
(3) Events and time are fundamental to the behavior of neural elements. DEVS naturally models event-driven processing and captures precise timing requirements at a high level, which can substantially improve design productivity and minimize the power consumption of implemented designs.
(4) Neural elements can be easily modeled by DEVS atomic models. Neural elements communicate with each other in networks that can be easily modeled via a DEVS coupled models.
(5) Stochastic representation is often required to accurately depict the behavior of neural elements. DEVS includes Markov models [1,26,27] that can be coupled with other DEVS components for design, modeling, and simulation.

1.2. Functional Architecture for Hybrid Intelligent Cyberphysical Systems

Figure 1 sketches the functional layers for a hybrid (discrete event, discrete time, and continuous modeling formalisms) control architecture for intelligent cyberphysical systems built under the DEVS system-theoretic framework [1]. An instance of architecture is the detailed design model of an unmanned vehicle and its associated discrete-event and continuous variable controllers. This simulation model was developed using a unified and flexible DEVS-based toolkit for modeling and simulation of a complete functional stack to develop and validate the system. Castro et al. [28] showed that the underlying DEVS simulator is capable of managing all dynamics (discrete event, discrete time and continuous) involved in the hybrid system. Differential equations represent continuous dynamics of the aircraft on the physical layer using quantized state system (QSS) integrators. A finite state discrete event controller at the top layer links to the action implementation layer containing a coupled model of DEVS atomics planners and conventional feedback controllers in. The DEVS-based architecture achieved simulation speed-ups of up to one order of magnitude above widely employed robot middleware-based simulation setups thus enabling on-board predictive simulations.

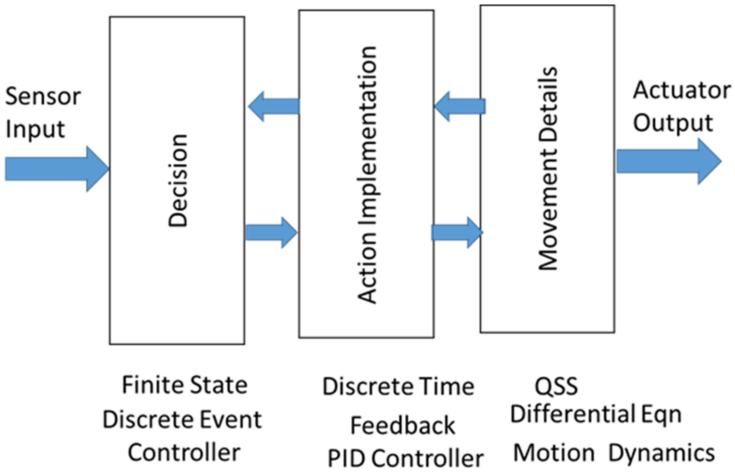

Figure 1. Functional Layers for Hybrid Intelligent Cyberphysical System Architecture.

Taking this architecture as representative of state-of-the-art DEVS-based hybrid intelligent cyberphysical system architectures, we will focus on the decision layer and review the architectural features presented in [11] in order to examine them for updating to current knowledge.

2. Methods

We now describe our methodology for developing DEVS building blocks and architectural patterns for Hybrid Intelligent Cyberphysical Systems.

Our approach to establishing that a model is a minimal realization of a given cognitive behavior, illustrated in Figure 2, is to specify the behavior as a system at the I/O behavior level. This requires us to define the behavior formally as a function mapping input segments to output segments (Figure 3). We employ structural inference to then seek a DEVS model at the state description level that generates the defined behavior. This amounts to associating the state description to the original I/O system description. This shows that the putative model is a realization of the specified behavior and then try to show it is a minimal realization. The state level description sets up the design of a network equivalent which amounts to a specification at the coupled component level of systems specification which we may be able to prove is homomorphic to the established minimal realization or we validate that it generates the required behavior by direct simulation. The network form is typically closer to potential plausible biological implementations and the established minimality of realization allows to claim that the latter realizations must be a homomorphic image of any such realization. (Please refer to [1] (Chapter 16) for a detailed exposition).

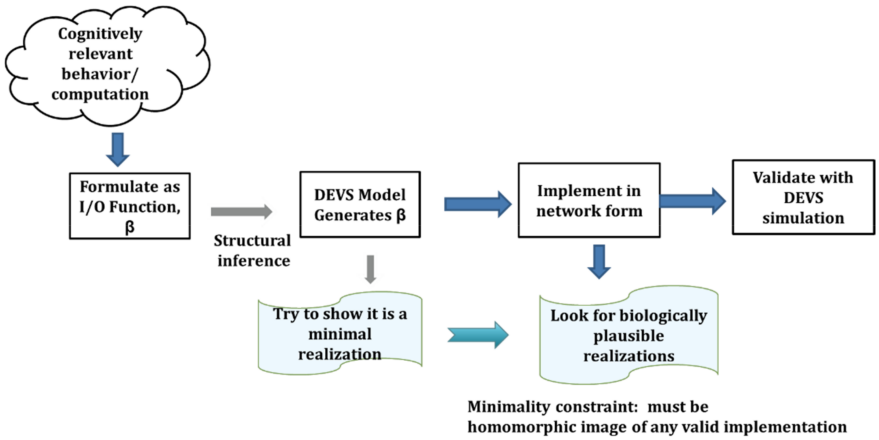

Figure 2. Approach to formulating behaviors and minimal realizations for building blocks.

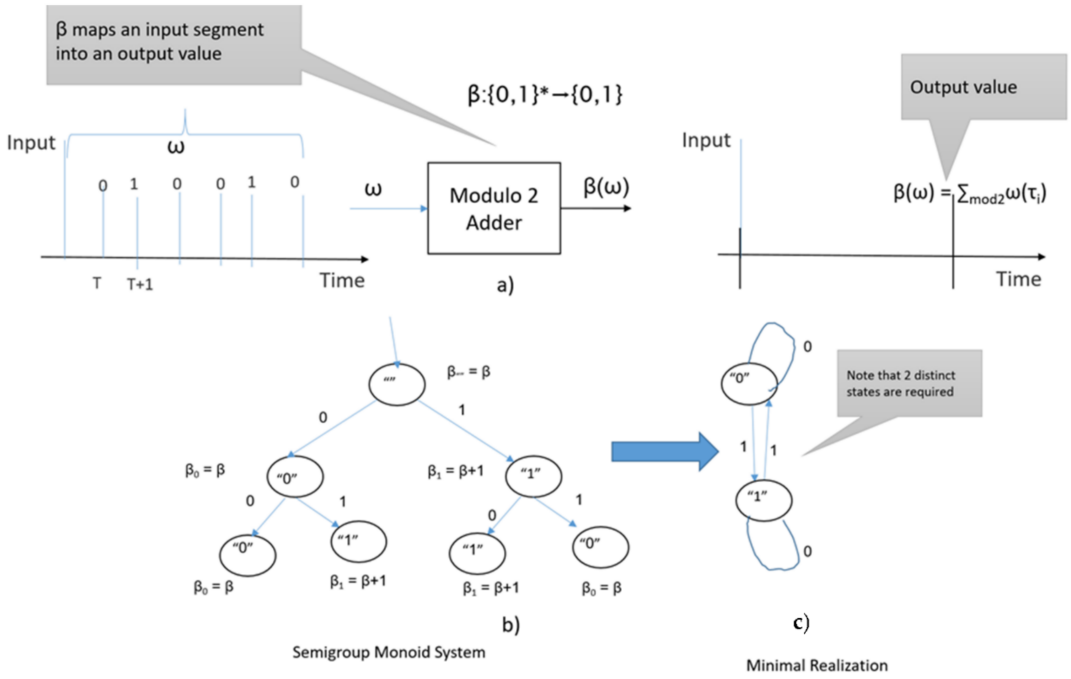

Figure 3. Example of the definition of an I/O function and deriving its minimal realization. The I/O function definition for modulo 2 addition is in (**a**), the semigroup monoid is in (**b**), and the minimal realization is in (**c**). * refers to Kleene star.

Using a modulo 2 adder as an example, Figure 3a, we briefly review the approach to deriving a minimal realization from an input time function description. The I/O behavior manifested by the adder is illustrated in Figure 3a as the function β:{0,1}*$\rightarrow$ {0,1} where

$$\beta(\omega) = \sum_{mod2} \omega(\tau_i). \quad (1)$$

Here, {0,1}* is the set of all strings with input set X = {0,1} and the function specifies the mod2 sum of the input string as output at the end of the input. Figure 3b shows how the minimal realization is derived in this case using the approach of constructing the reduced version of the monoid (semigroup with identity) representation of the set of all strings. As an example, X = {0,1} and starting with the empty string "", the tree grows by adding branches for inputs 0 and 1 respectively. Whenever a node is encountered with the same derivative function as one earlier encountered it can be merged with the earlier one. For a finite state function this will eventually bring a halt to the unfolding of the tree. Here we find that the output from node "0" is the same as from node "" and that the output from node "1" is different. This shows that we cannot merge these nodes. Furthermore, expansion of the definitions for the derivative functions, $\beta_0(\omega) = \beta(0\omega)$ and $\beta_1(\omega) = \beta(1\omega)$, shows that these are equal to $\beta(\omega)$ and $1 + \beta(\omega)$, respectively. This implies that continued unfolding of the tree will not yield any new derivative functions. Thus, the minimal realization has two states corresponding to the two distinct nodes with transitions reflecting the alternating pattern exhibited in the tree.

Homomorphisms [29] are useful in proving that implementation are correct realizations and establishing minimal realizations. The general concepts of homomorphism and isomorphism relate system models at the same level of specification. Corresponding to each of the system specification levels is a relation appropriate to a pair of systems specified at that level. We call such a relation a preservation relation or system morphism because it establishes a correspondence between a pair of systems whereby features of the one system are preserved in the other. Morphisms appropriate to each level of system specification are defined such that higher level morphisms imply lower level morphisms. This means that a morphism which preserves the structural features of one system in another system at one level, also preserves its features at all lower levels.

Indeed, the minimal realization can be shown to be a homomorphic image of any realization of its behavior. In particular, in this case the function β is a homomorphic mapping from the monoid system of Figure 3a to the minimal realization. There is a close relation between homomorphisms and implementations in network form. For example, the Krohn-Rhodes theory of finite state machine decomposition shows that any such system can be homomorphically realized by a cascade of basic primitives that themselves are not decomposable [30]. As a special subcase, any module m counter can be realized by a cascade of prime counters. For example, a four-counter can be implemented in this way by a pair of binary counters of the form in Figure 3b. In this paper, we are seeking such building blocks, primitives, and compositions useful for intelligent hybrid cyberphysical system design. Stemming from this account, we refer to building blocks as models that can be replicated and reused in different standardized configurations called architectural patterns where a building block may be called primitive if it cannot be decomposed further.

Figure 4 illustrates a DEVS hierarchical coupled model that provides the proposed basic ontology of terms for components and their interactions that an intelligent system such as a neuron-level architecture of the brain would contain. Rather than pointing to neurons as the main components, we refer to Neuron Elements (NE) which stand for the components that perform the primary processing functions typically associated with neurons but that may include other supporting elements beside neurons. Similarly interconnect elements (IC) generalize the role of synapses in connecting information flows among the NEs. The architecture may be hierarchical in that the main components NE and IC are in fact coupled models themselves. Considering the information flows themselves leads us to examine the form of messages sent/received by the components and the corresponding types of the inputs and output ports. As examples of such considerations,

neuromorphic hardware implementations of spiking neuron nets have to make such architectural choices in order-to-tradeoff processing capability with energy consumption and design complexity [1].

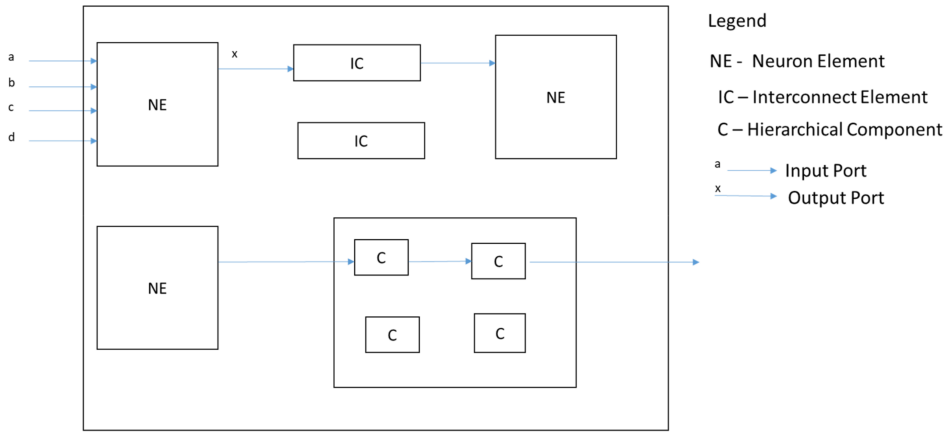

Figure 4. DEVS Hierarchical Coupled Model that illustrates the terms to be used.

3. Results: Building Blocks and Architectural Patterns

In this section, we show how the methodology presented is applied to obtain DEVS building blocks and architectural patterns for Hybrid Intelligent Cyberphysical Systems.

3.1. Fast Discrete Event Neuron Architectures for Decision Layer

We proceed to review the DEVS characterization of neural network models based on discrete rather than continuous abstractions. DEVS abstractions capture the many features of biological neurons that are not represented in conventional artificial neural networks and to exploit these capabilities to perform intelligent control tasks [31,32]. Of particular note is the "One-spike-per-neuron" concept that refers to information transmission from neuron to neuron by single pulses (spikes) rather than pulse trains or firing frequencies [33]. 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. "Act-as-soon-as-evidence-permits" behavior can be implemented by "order-of-arrival" neurons which have plausible real-world implementations [33]. 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.

The basic concept that supports discrete event abstraction 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. 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 combinatorically more efficient information code than one based on the number of stimulated input wires rather than their order of stimulation [33].

The "one-spike-per-neuron" architecture [33,34] proposed several processing layers between input from sensors and outputs going to actuators. Sensory layer neurons react directly to incoming energy (in various modalities such as visual or infrared electromagnetic waves, sonar, etc.) These neurons perform the analog-to-latency coding just discussed. Fusion/Analysis layer neurons fuse the data collected from the various sensors into stereotyped complexes 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.

In [11] it was hypothesized that a proposed generic neuron model could be specialized to satisfy all the requirements for behavior at each of the layers. Here we step back and examine the problem of uncovering fundamental requirements, whether implementable in a single generic neuron or not, for realizing cognitive behavior that are also biological plausible.

3.2. Fast and Frugal Heuristics: Voting Example

The "fast frugal and accurate" (FFA) perspective on real word intelligence [12,13] provides a holistic cognitive system framework that characterizes decisions that an intelligent system has to make under time pressure and limited information. FFA hypothesizes that fast and accurate heuristics can be constructed from simple building blocks that control attention to informative cues, quickly terminate search processing, and make final decisions. 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. Consequently, it is non-compensatory, non-linear, and can violate transitivity, the cannon of rational choice.

As an example of FFA, we review a recent study of how people determine their votes for political candidates [14]. Recognizing that the actual amount of political information possessed by voters is small, unevenly distributed, and time pressured, the study confirmed the predictions of [14] which predicts that employing specific cognitive shortcuts can help voters make good and quick decisions, even with a lack of information. Voters appear to simplify decision making by focusing mainly on those policy positions they consider important. The study measured the congruence between a voter and candidate by the distance between their positions on the most important issues to the voter. This enables the assertion that voters' decisions are correct if they have chosen the candidate with the highest congruence. This explains experimental results where voters are able to make correct decisions (as measured in this manner) independently of the amount of information available. The results confirm that fast and frugal heuristics may be the decision strategies that voters adopt to make quick, frugal, and correct (as measured) decisions.

"Take the best" amounts to using a lexicographical ordering of candidates based on importance of issues and matching between a candidate's and voter's views on the issues. In other words, the voter looks at their most important issue and compares candidates on this issue. The candidate most congruent to his views on this issue is chosen unless it is not possible to rank on this issue, in which case the same procedure is applied to the next important issue, and so on. Note that importance of issue does not depend on the candidate—so importance comes first in the lexicographical order.

If fast and frugal heuristics characterize human bounded rationality, then they ought to be implementable by discrete event neural architecture introduced earlier. Indeed, let us sketch a construction to show this to be the case. Figure 5 depicts a network of DEVS components that implements "take the best" for the voting example. First Arrival is an NE that lets the first spike to arrive through and blocks subsequent arrivals. Computation

Delay is an IC between First Arrivals that delays an incoming spike by an amount specified by a parameter. As a model of cognitive behavior, the delay represents in an inverse manner the strength of the weight of the connection—the shorter the delay the earlier the spike will arrive at the target node. Thus, in Figure 5, the Computation Delays represent the respective importance of issues to the voter so that the most important issue emerges from the First Arrival en route to the downstream instance. These delays are packaged together in a Parallel Delay that is a hierarchical parallel DEVS coupled model. The most important issue continues to be evaluated in parallel for each candidate with the one most congruent with the voter's attitudes emerging from the second First Arrival.

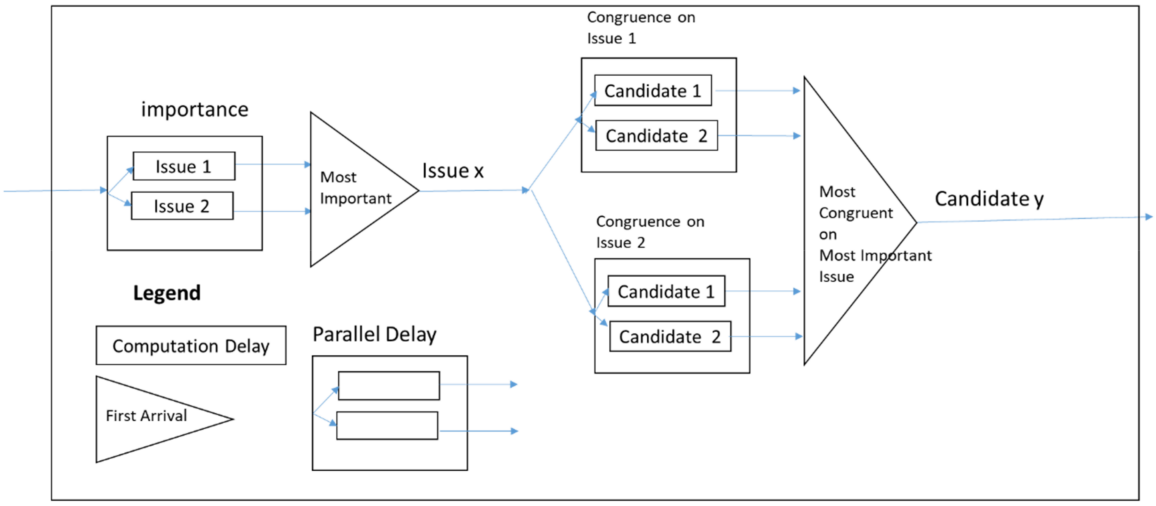

Figure 5. DEVS neural architecture implementing "take the best" inferencing.

Figure 5 identifies potential building blocks with First Arrival and Computation Delay as primitives and Parallel Delay as a composite. We focus on First Arrival to consider its minimal realization and potential implementations. Recall that our approach to establishing that a model is a minimal realization of a given cognitive behavior, illustrated in Figure 6, is to specify the behavior as a system at the I/O behavior level. This requires us to define the behavior formally as a function mapping input segments to output segments.

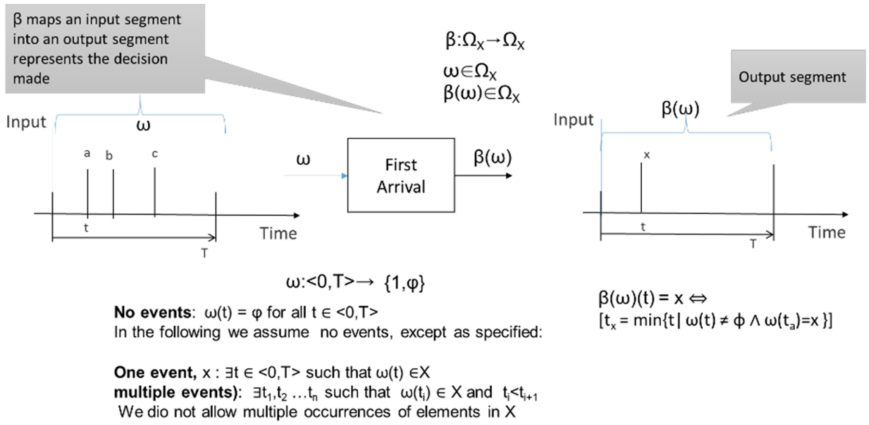

Figure 6. Formal representation of first arrival via definition of an I/O function.

The I/O Behavior manifested by the First Arrival is illustrated in Figure 6 as the function

$$\beta{:}\Omega_X \rightarrow \Omega_X$$

where Ω_X is the set of all DEVS segments with input set X and for all $\omega \in \Omega_X$, $\beta(\omega) \in \Omega_X$. β is defined by

$$\beta\ (\omega)(t_x) = x \Leftrightarrow [t_x = \min\{t \mid \omega(t) \neq \varphi \wedge \omega(t) = x\}] \quad (2)$$

where t_x is the time at which the event with value x occurs. This states that the only output, if any, is the event occurring first in the interval. Note that the definition also requires that any events occurring after the first do not appear in the output.

Figure 7 shows how the minimal realization is derived in this case using the approach of constructing the reduced version of the monoid (semigroup with identity) representation of all DEVS segments as generalized from the case of strings introduced above. As an example, X = {a,b} and starting with the empty segment the tree grows by adding branches for segments ending with events a and b, respectively. Whenever a node is encountered with the same derivative function as one earlier encountered it can be merged with the earlier one. For a finite state function this will eventually bring a halt to the unfolding of the tree. The output function requires outputs with these respective values and that subsequent inputs do not result in outputs. A null event segment does not change the output. Thus, there are 4 different derivative functions β, β_a, β_b, and β_{aa} for the original requirement, β, the outputs after the input events a and b, respectively, and the blocking of subsequent input. In general, we see that n + 2 states are required for an event set X with cardinality n.

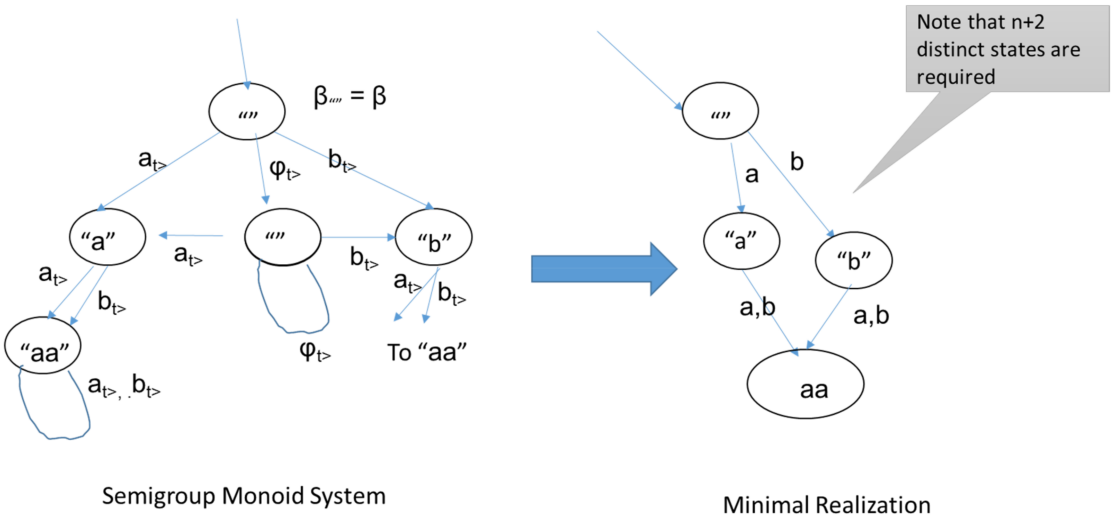

Figure 7. Semigroup monoid system representation of first arrival behavior and minimal realization.

The DEVS model specification for this behavior can be presented as the following atomic model:

$$M = <X, Y, S, ta, \delta_{int}, \lambda>$$

where

X is the set of input values = {a,b}
Y is the set of output values = {a,b}
S is the set of partial states of the system = {waitforInput,send,passive}
ta: is the function of advancing time:
ta(send,x,0) = 0
ta(waitforInput,null,∞) = ∞
δ_{int}: is the internal transition function:
δ_{int}(send,x,0) = (passive,∞)
δ_{ext}: is the external transition function:
δ_{ext}(waitforInput,null,∞,e,x) = (send,x,0)
δ_{ext}(send,x,0,e,x′) = (send,x,0)—ignore later arriving input

δ_{ext}(passive,∞,e,x) = (passive,∞)

λ: the output function:

λ(send,x) = x

Note that the specification is deceptively simple as it hides the underlying semantics that operationalize the definition in the proper simulation infrastructure. Please refer to [1] for a detailed exposition.

Figure 8 displays a user-developed state diagram in MS4 Me [27] for this DEVS model.

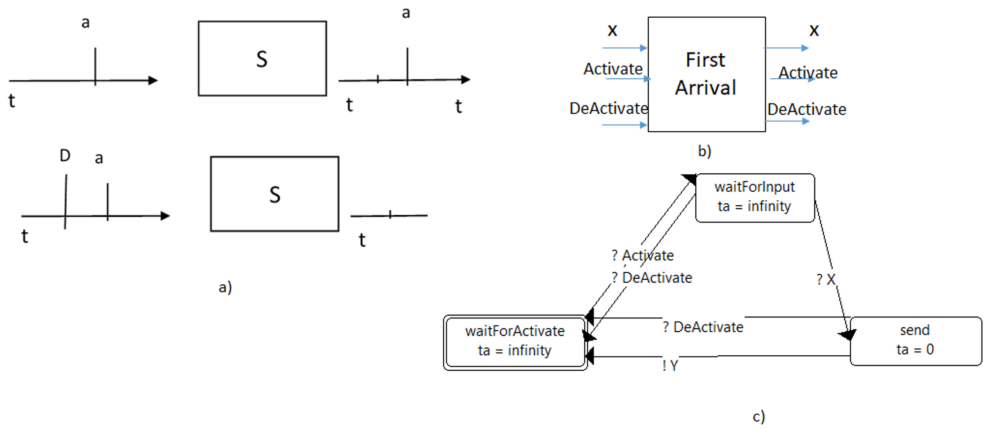

Figure 8. De-activate behavior (**a**) and implementation of activation in in FirstArrival (**b**,**c**).

To establish a homomorphism from this model to the minimal realization, we note that the states waitforInput and passive map to the initial and final states of the latter. The state send, is actually the phase of the state pair (send,x) where x is the event received so that it represents the storage required for the n states of X. We can easily trace the transitions of this model and of the minimal realization to check that the correspondence is preserved.

3.3. Activation as Distinct from I/O Processing

We now introduce an important capability called *activation* that is distinct from input/output processing. As illustrated in Figure 8, the state waitforActivate replaces the waitforInput and passive states as the starting and ending state. An Activate input sets the model to a new waitForInput state that is equivalent to the original except that it can be sent back to the initial state by a DeActivate input. The representation in DEVS is obtained essentially by adding the definitions:

δ_{ext}(waitForActivate,∞,e,Activate) = (waitForInput,∞)

δ_{ext}(waitForInput,∞,e,DeActivate) = (waitForActivate,∞)

δ_{ext}(send,∞,e,DeActivate) = (waitForActivate,∞)

While appearing to be a small change when represented in the state diagram, the additional capability has large consequences in the implementations to be discussed.

3.4. Variations on First Arrival

As a first example where activation facilitates design, we consider detecting the arrival of the first two candidates.

A pair x, x′ will be the *earliest arriving pair* if

$$\beta(\omega)(t_x) = x \wedge \beta(\omega)(t_{x'}) = x' \Leftrightarrow$$

$$[t_x = \min\{t \mid \omega(t) \neq \varphi \wedge \omega(t) = x\}] \wedge [t_{x'} = \min\{s > t \mid \omega(s) \neq \varphi \wedge \omega(s) = x'\}] \quad (3)$$

To implement the ordered pair arrival, we can employ a coupled model of FirstArrival units with activation capability as in Figure 9.

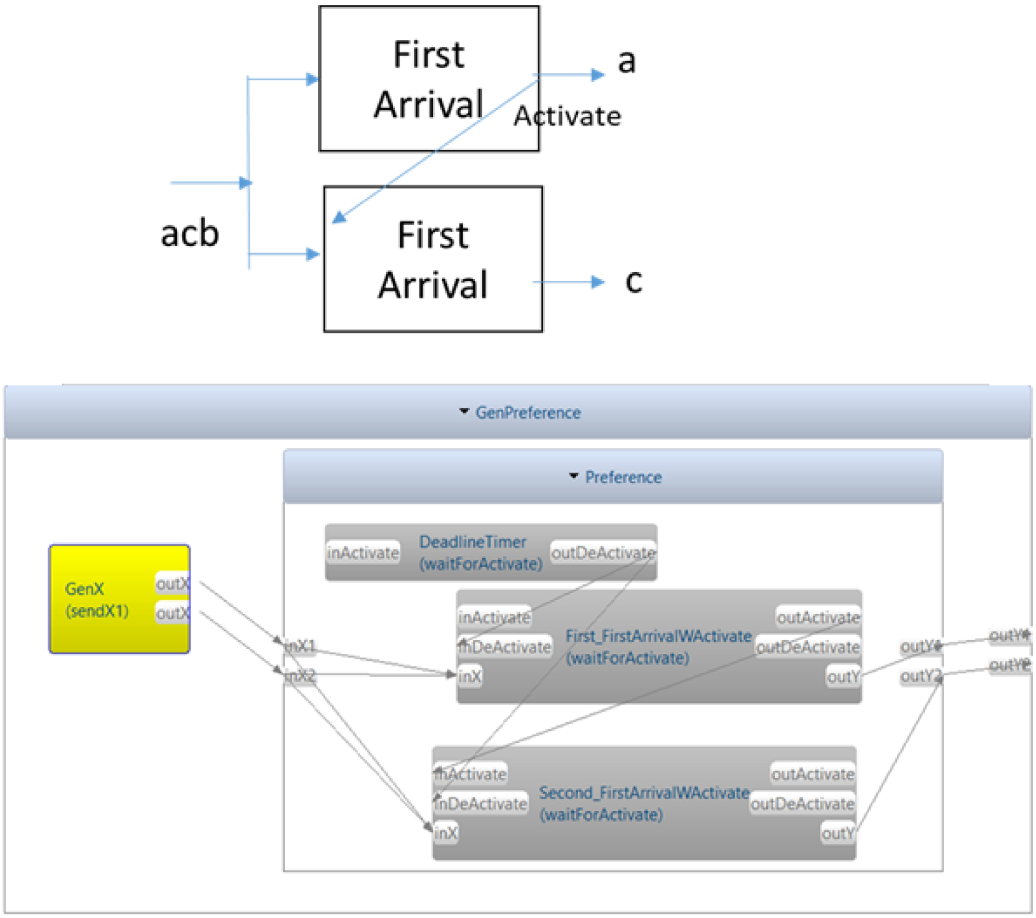

Figure 9. Coupling of FirstArrival models and implementation in the MS4 Me environment [26].

Here the input is sent to both components as in a parallel composition but the Activate coupling is implemented in a serial manner. The upper FirstArrival starts in the activated state and detects the first arrival while the second component is not active. Receiving the activation from the first (now passive) component, the second component can now wait for the second arrival and respond to it when it arrives. Any subsequent inputs will be blocked by both units. Note: we will discuss the included DeadlineTimer component momentarily.

Following the methodology of Figure 2, we ask how the realization in Figure 9 relates to a minimal realization of the ordered pair arrival behavior. By unfolding the monoid tree as earlier discussed, we note that the tree expands to depth 2. Due to the restriction on reoccurrence, the number of nodes beyond the root is $n(n-1)$ where n is the cardinality of the event set. Therefore, the required number states is $n^2 - n + 1$. In comparison, the pair of FirstArrival components gives the coupled model a state cardinality of $(n+2)^2$ so that it has order(n) more states than minimally required. As before, we can define a homomorphism from the FirstArrival pair to the minimal realization noting the first level of the tree maps to the first unit states with a similar mapping from second level to the second unit.

Note that this method can be extended from ordered pairs to ordered tuples of any size, m. The coupled model of Figure 9 then expands to contain m FirstArrivals with the same kind of coupling pattern which distinguishes I/O processing from activation—the former a distribution in parallel, and the latter in serial. The implementation has $(n+2)^m$ states while the minimal realization has $n(n-1)\cdots(n-m+1) = n!/(n-m+1)!$ states. Again, a homomorphism from the implementation to the minimal realization maps successive stages of the implementation to successive levels of the monoid tree.

Note that Equation (3) requires the output function to reproduce the order in which the pair of events occurred. If the order does not matter to the downstream processing than the unordered pair would be defined as output. In this case, a function γ would be defined that emitted the unordered pair {x, x'} at the time of the second arrival:

$$\gamma(\omega)(t_{x'}) = \{x,x'\} \Leftrightarrow [t_x = \min\{t \mid \omega(t) \neq \varphi \wedge \omega(t)=x\}]$$

$$\wedge\, [t_{x'} = \min\{s > t \mid \omega(s) \neq \varphi \wedge \omega(s)=x'\}] \quad (3)$$

As before this definition can be extended to any size, m of arriving tuple so that the unordered set of the first m events that arrive is output. This reduces the size of the minimal realization to the combinatorial $n!/(n - m + 1)!m!$ states. We note that the same form of implementation can be retained, leaving the question open of constructing a smaller implementation that exploits the loss of information due to removal of order in the output.

Another possible variation is to set a deadline and accept only those events that occur before the deadline instant, D:

$$\beta\,(\omega)(t) = x \Leftrightarrow [t < D \wedge \omega(t) \neq \varphi \wedge \omega(t) = x]. \quad (4)$$

As illustrated in Figures 9 and 10, an implementation here can add a timer in parallel with a FirstArrival composition and have it deactivate the latter at the expiration of the deadline duration. Given that no limitation is placed on the occurrence of the events relative to the deadline, the minimal realization must still be able to reproduce the original output and there it must have at least the same number of states as before. In addition, the composition with the timer, which has initial and final states, doubles the number via the cross-product.

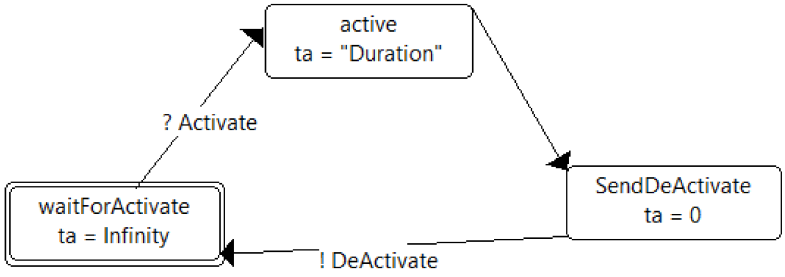

Figure 10. Timer for implementation of deadline-based decision.

3.5. Perception of Order of Discrete Event Arrival

With an ultimate interest in generation and recognition of DEVS segments of events we first characterize the behavioral requirements underlying perception of order of discrete event arrival. We will provide a minimal realization for such behavior and use its implementation subsequently in the finite language framework for DEVS segments. As in Figure 11, consider two discrete events, a and b, that occur in a given time interval and are input to a system that is supposed to output judgements about the arrivals in relation to each other. We represent the perception of order of occurrence as a function that maps event segments into a value at end of the interval. Further, different from the considerations earlier made for decision making, we include the separation in time between the events as relevant. We start with the requirement that the function be able to mark the arrival of b after a within a given interval. Other possibilities to distinguish include a or b happening alone, simultaneous occurrence, and no events occurring within a given interval. As in Figure 11, we formulate the computation, β, as an I/O Function behavior in the DEVS formalism. We then provide a realization at the state level as a DEVS atomic model and show this to be a minimal realization.

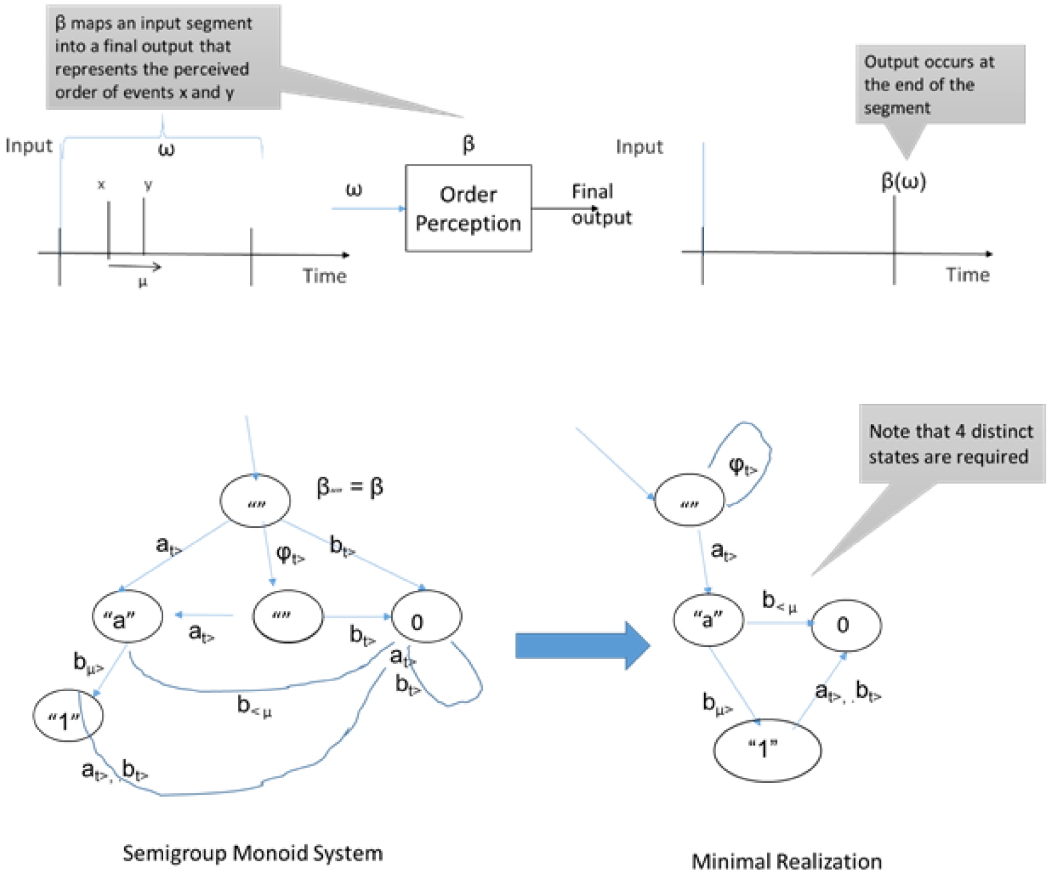

Figure 11. Order of arrival behavior definition and minimal realization.

The definition of the function becomes:

$$\beta(\omega) = 1 \Leftrightarrow \exists s,t: s< t < s + \mu \wedge \omega(s) = a \text{ and } \omega(t) = b$$

$$\beta(\omega) = 0 \text{ otherwise} \quad (5)$$

Unfolding the monoid tree, we obtain Figure 11 which leads to the minimal realization. Note that we distinguish between an arrival of event b following event a within interval μ and later than that by corresponding segment length limits.

As above, we can formally represent the behavior of the order-of-arrival neuron mentioned earlier. By unfolding the monoid tree, we can lay out a minimal realization which given any specific order of arrival of n distinct events, i.e., any permutation of the events, recognizes it with n+1 states. By unfolding the tree, we see that the minimal realization of the behavior that recognizes any one of an incident permutation of n events has n! + 1 states (a reduction in size from n^n states required to recognize any combination of n events).

3.6. Elementary Perception Unit

To create an Elementary Perception Unit (EPU), the FirstArrival design is extended to include a parameter specifying a particular event to recognize and a time to wait for the event after activation. As in Figure 12, this makes use of the DEVS time advance function that assigns a time to remain in a state before transitioning to the next assigned one—in this case to a state indicating that no event of the specified form has been received.

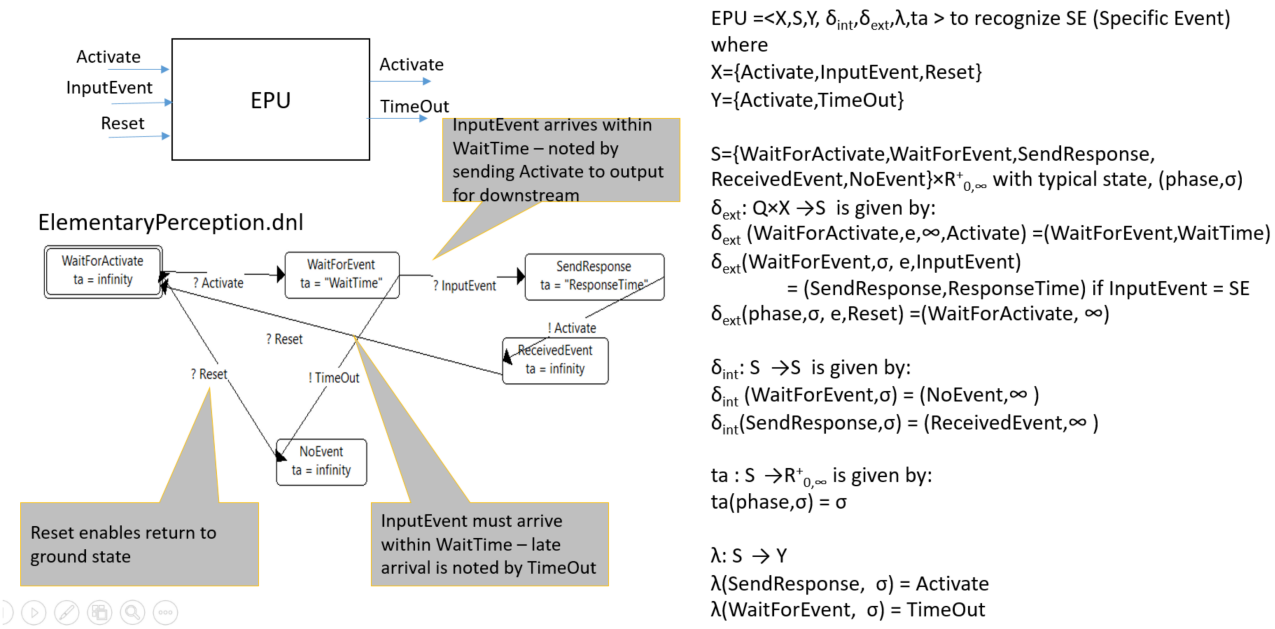

Figure 12. Definition of Elementary Perception Unit (EPU).

The coupled model in Figure 13 implements the order of arrival of event a followed by event b, where the EPUs recognize a and b, respectively. The coupling manifests the earlier pattern in which activation and I/O processing couplings are distinguished. Activation is passed on sequentially from the first EPU to the second upon the positive event recognition by the latter. In contrast, the input is distributed to all components uniformly, as is the reset input. Similarly, activation is emitted externally only from the last EPU. However, the time out notice can emerge from any component.

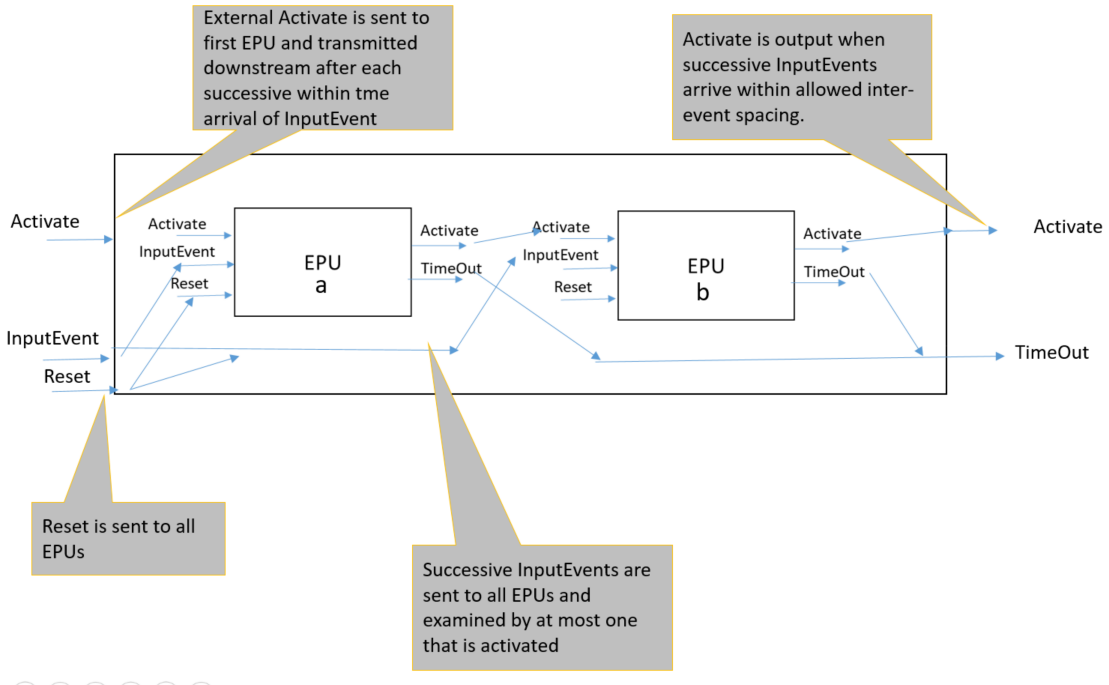

Figure 13. Series coupling of EPUs showing difference between input and activation coupling patterns.

In the following we extend this capability to regular languages which are sets of event segments that can be defined in a generalized finite state manner.

4. Finite State Regular Language Framework

In [10], we identified a finite state regular language framework as a basis for modeling of cognitive behavior. In brief, regular expressions are strings of symbols that include the alphabet and connectors for concatenation, Boolean operations, and Kleene star (denoting iterative concatenation) [35]. we proved that given a temporally represented finite alphabet and a regular expression over the alphabet, we can construct a DEVS coupled model that constitutes a networked recognizer for the regular expression.

Regular expressions are descriptions of behavior at the I/O Behavior level for which there are computable realizations by finite state automata. Indeed, for every regular expression there is a finite state acceptor (that can recognize strings in its language) and a finite state generator that can generate such strings [36,37]. Therefore, we took this framework as the basis for the computations that mediate between cognitive and neural domains, and we identified realizations of such system specifications at the coupled system level involving neural componentry and potentially other needed cellular elements.

4.1. DEVS Recognizer Models

Let DEVS(X) be a set of DEVS segments based on X a finite alphabet. Given an input segment,

$$\varphi_{t1} > x_1 \varphi_{t2} > x_2 \ldots \varphi_{tn} > x_n$$

We strip the time information to obtain the string $x_1,x_2,..,x_n$. The erasure of time information is called the natural encoding of DEVS segments to strings. We extend the concept of language recognition and generation from sequences to DEVS segments. Roughly, this requires that the DEVS model partitions the set of all segments into accepted and non-accepted sets just in case the stripped sequences are partitioned in the same manner by an automaton. Indeed, the way we proceed is to extend a finite state automaton to a DEVS in such a way that the language it accepts is in fact, the set of sequences stripped from the DEVS input segments. Figure 14 illustrates the approach. An atomic model that recognizes a single symbol e.g., the language {x} appears in Figure 14a with input Activate to send it from the Wait state to the Hold State as shown in Figure 14c. After this, an input with label x happening some time later will cause the model to enter the Send state and to output an Activate output signaling the acceptance of the input. Any other input label will return the model to the Wait state without sending such an output. The model in DEVS set notation appears in Figure 14b.

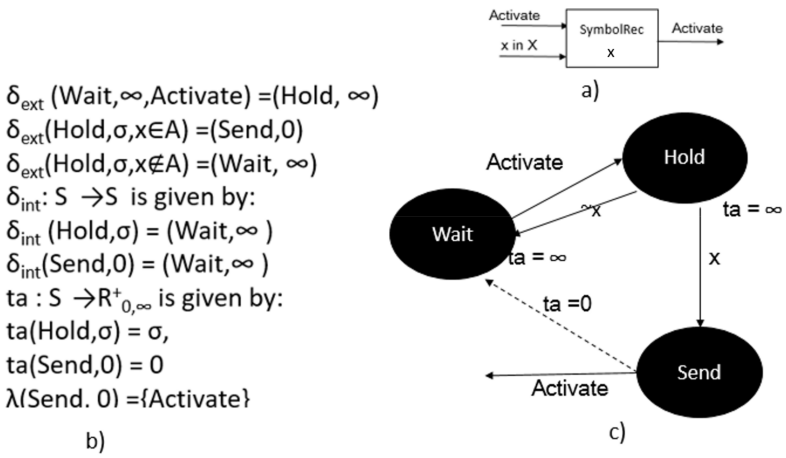

Figure 14. DEVS recognizer model. The interfaces are shown in (**a**), the model specification is in (**b**), and the state diagram is shown in (**c**).

4.2. Constructing a DEVS Recognizer for Segments Sets in DEVS(X)

As illustrated in Figure 15a, we define an input-driven DEVS as a DEVS model for which the time advance in all states is infinite. Effectively, this means that the only source

of state change is from external events and we may omit the internal transition function as in the definition: M = <X,S,δ_{ext},ta> where ta(s) = ∞, for all s ∈ S. For convenience we also omit the output function. Furthermore, we will require that the elapsed time, e plays no role in state transitions due to inputs so that: δ_{ext} (s,e,x) = δ(s,x). Now, as illustrated in Figure 15b, a transition system, underlying a finite automaton, can be defined as T = <X,S,δ> where δ:S × X→ S. Therefore, we have the following proposition:

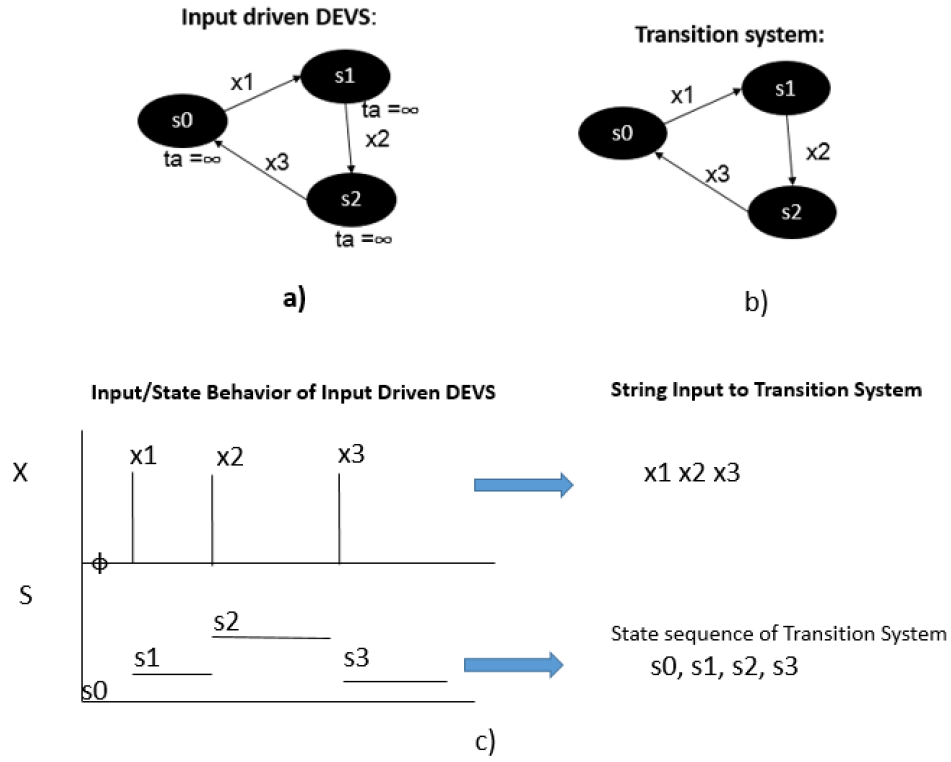

Figure 15. Mapping from input-driven DEVS to transition system.

Proposition 1. *Given a transition system, T, there is a DEVS M that homomorphically simulates it.*

A sketch of the proof is illustrated in Figure 15c. Construct a DEVS M whose states and transitions are labelled in one-one correspondence to those of T. Suppose that T undergoes a state sequence s0, s1, s2, s3 in response to an input sequence x1, x2, x3. Then if M is fed an event segment with natural encoding to x1 x2 x3, M will undergo the state trajectory shown which in stripped form matches that of T. Technically, we can show that the natural encoding together with the one-one state mapping constitute a homomorphic simulation of T by M (see definition in [1]).

We also readily have the converse:

Proposition 2. *Given an input-driven DEVS, there is a state transition system that it homomorphically simulates.*

Now we can extend T to an automaton A_T by adding to it labels for an initial state and a set of final states. Likewise, extend M to a DEVS Recognizer, R_M with addition of the same initial state and set of final state labels. Then stated in succinct form:

Corollary 1 *The language accepted by $R_{M.}$ is also accepted by A_T.*

Expanded, this means that the set of DEVS segments that drive the input free DEVS from its initial state to a final state, when stripped of their timing, is the set of sequences that drive the corresponding automaton from its initial state to one of its final states.

Note that to conform to the input/output interface for the DEVS model of Figure 14a, we have the model start in a Wait state and have it transition to the initial state upon

receiving an Activate output. Likewise, we have it generate an Activate output when transitioning into a final state. This converts its input-free nature to a full-fledged DEVS. However, this interaction is only manifest for the Activate interface.

Theorem 1. *A subset of DEVS(X) is recognized by an input-driven DEVS if, and only if, the natural encoding of the set is a regular language. We call such a subset of DEVS(X) a regular language of DEVS segments.*

The proof centers on the fact that the internal operation of an input-driven DEVS is exactly replicated in the automaton constructed from it. Conversely if a set is accepted by an automaton, then an input-free DEVS can be constructed that replicates this behavior in relation to the natural encoding.

4.3. Regular Realization at the DEVS Network Level

In this section our goal is to develop templates for realizing the behavior expressed by regular languages in the form of DEVS coupled models. This will be done by realizing finite state automata in network form in a particular way that allows generalization to continuous time segments.

There are multiple representations of a regular language as a regular expression and as a finite state automaton. The mapping in Figure 16a,b gives a simple example. Further there are multiple realizations of finite state machines in networked form as e.g., well-known clocked sequential circuits. As illustrated in Figure 16b,c, the realization approach we take employs the recognizers for elementary symbols from Figure 16. In the mapping, each transition label is mapped to a recognizer of that label and each state is mapped to an activation node shown as a black oval component. For example, the states s0 and s1 are mapped to activation nodes s0 and s1, while the transition from s0 to s1 labelled a is mapped to a recognizer for a. Starting in initial state s0 is represented by external activation of its representative component. External input is coupled to each recognizer's input port. As in Figure 12, a recognizer can only pass on activation to its target state representative if has received activation from a predecessor state. The mapping is algorithmic and can be implemented to generate coupled model recognizers for the source regular expression as illustrated in Figure 16d in a DEVS simulation environment [26].

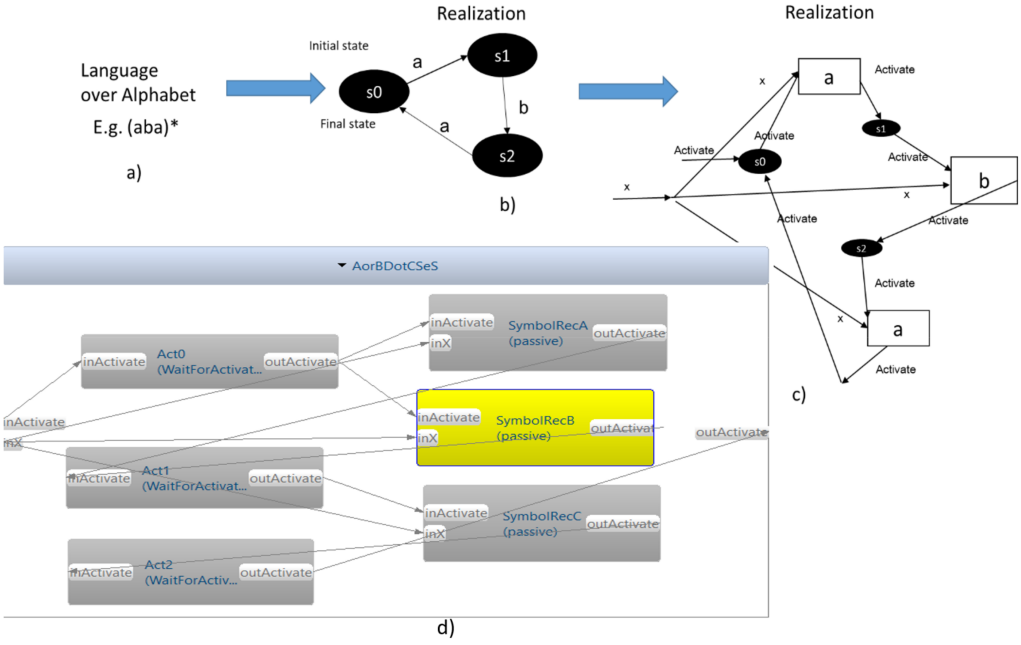

Figure 16. Realization of Regular language at Finite State and Network levels. An example language is in (**a**), its finite state acceptor is in (**b**), the mapping to network form is in (**c**), and the implementation is shown in (**d**).

Note that the activation node behavior can be easily captured by a simple atomic DEVS which produces an Activate output in response to the first Activate input that it receives. Quick considerations shows that the minimal realization of this behavior requires only two states.

Formally, we can prove the theorem:

Theorem 2 *A regular language of DEVS segments can be realized by a coupled model of symbol recognizers and activation nodes.*

The proof shows that exactly one activation node is in the Active state and it enables the recognizers for its corresponding state's outgoing input labels. Only one recognizer for each label accepts the next external event (assuming deterministic automata) and activates the target activation node representing the next automaton state. The proof then proceeds by induction to show that at the end of the input segment a final state is reached or not according to whether or not it is in the regular language.

5. Discussion

In this article, we described and applied a methodology for developing DEVS building blocks and architectural patterns for Hybrid Intelligent Cyberphysical Systems. Before discussing potential applications, we provide a summary of the models discussed.

5.1. Summary of Building Blocks and Architectural Patterns

Building blocks can be summarized as:

- FirstArrival: used to select the first (strongest)competitor to produce a proposed response
- Computation Delay: converts strength of proposed responses to speed of travel to First Arrival selector
- FirstArrival Variants: can sub-select contenders for further down-selection
- Elementary Perception Unit: recognizes arrival of event
- Elementary Generation Unit: generates events in time
- Activation Units: transmit activation in networks

Architectural patterns:

- FFA networks: First Arrivals and Computation Delays to make informed choices
- DEVS-segment regular language generators and acceptors: implement finite state automata in network form of EPUs, EGUs, and activation units.

5.2. Multiple Subsystems Competition Application

Now we discuss an application that illustrates the application to system design.

An example where the overall methodology illuminates the modeling of cognitive behavior is offered by the multiple memory systems theory [38] which asserts different kinds of information are processed and stored in different parts of the brain. This can lead to different proposals for action in response to a situation and the need to resolve the competition. In Figure 17, the Procedural, Emotional, and Declarative systems operate in parallel with common inputs and outputs. Further, each have their own structures and independently extract different kinds of information from the ongoing activity. Procedural memory refers to constant stimulus-response (S-R) relationships that lead to successful outcomes. Information processed and stored in this system tends to produce the response whenever the stimulus is encountered. The Emotional system refers to (S-Af, or stimulus-affect associations) while the Declarative system stores and applies S-S higher level cognitive factual knowledge relations.

An experiment [38] tested the theory's claim of independent parallel potentially conflicting decisions by placing rats in a maze with different configurations to elicit different foraging strategies.

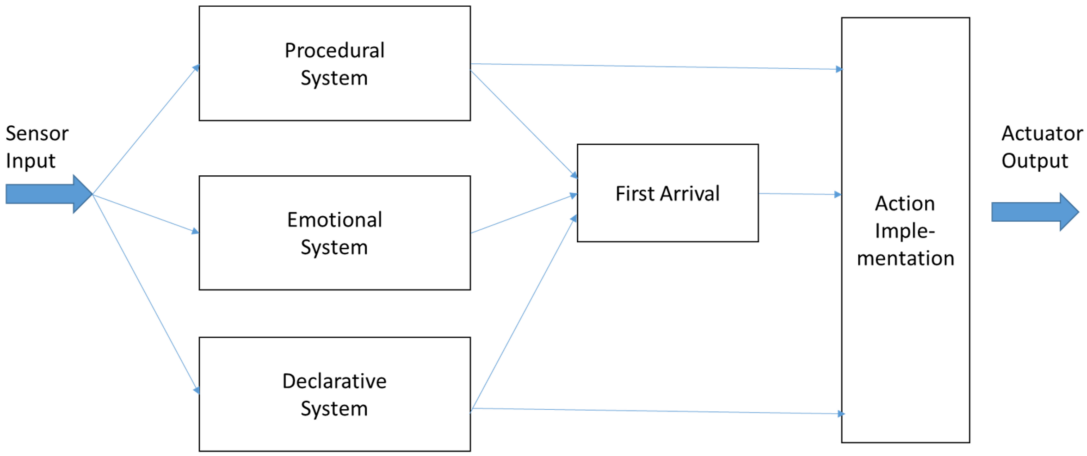

Figure 17. First Arrival-based selection of actions vs. resolution of conflict at the output stage.

Each configuration provided the rat with a different kind of information about the location of food on the maze. Normal rats learned to locate the food easily in all three configurations. The first task requires keeping track of the locations of past successful accesses to food to enable predicting future food locations (requiring spatial map declarative processing). In the second task lights inform the rats about the current location of the food and they learn to associate the light with the food availability (requires operant stimulus-response procedural processing). The third task confines a rat to one location and requires associating environmental cues there with presence or absence of food (requiring emotional system classical conditioned response processing). Lesions to different parts of the brain confirmed the separate location and independence of the respective systems responding to these tasks. However, training from the first task did cause reduced performance in the second task as rats showed uncertainty whether to follow the light cues or the dependence on prior location.

To explain this reduction, the experimenters hypothesized that (1) the system with the most coherent representation produces the strongest output and wins the competition for control of behavior, but (2) another system with a less coherent representation can produce output that is sufficiently strong to negatively influence behavior [38]. In Figure 17, we show this direct resolution of conflict by direct outputs from the parallel systems to the actuator output. In contrast, the candidate selection computation primitive, represented by the FirstArrival component, would enable the conflict to be resolved before any output is generated. In this design, the relative strengths of the system outputs would be manifested as computation delays with the fastest gaining control of the final output. In a variation, the deadline feature could leave a subset to compete for final output with resulting degradation in performance. This would predict that under time pressure, output behavior is more likely to be optimal than with none applied. As in FFA, if rats are forced to make a choice, they make the best available. Thus, design of artificial agents might potentially improve on natural systems with more advanced designs for conflict resolution decision systems.

Future research should be focused on using and extending the methodology for further exploring potentially useful DEVS building blocks and architectural patterns for Hybrid Intelligent Cyberphysical Systems [39,40].

A note about using Classic DEVS rather than PDEVS as basis for the theory developed in this paper: Classic DEVS was used in an effort to keep the exposition as simple as possible for the reader unfamiliar with DEVS. I believe that the main results carry through to the PDEVS case which introduces the complications of simultaneous events and multiple simultaneous inputs. However, establishing this claim is left for future work. It should be stated that the current results apply to discrete event neural nets due to the vanishing probability of synchronous events [24].

6. Conclusions

Among the contributions of this research, this article provides the first attempt to uncover DEVS-based building blocks and architectural patterns for intelligent hybrid cyberphysical system design. We argued that discrete event abstraction captures information in rich continuous data streams in terms of events and their occurrences in time. 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 cyberphysical systems with intelligent cognitive behavior. More generally, any cyberphysical system must operate within the constraints imposed by space, time, and resources on its information processing. The discrete event paradigm provides the right level of abstraction to tackle these issues. Therefore, it is appropriate to seek DEVS-based models that can be employed in a repeatable manner to implement characteristic behavior requirements of smart systems.

More generally, DEVS may be the best candidate to implement conflict resolution in today's complex systems of systems design problems. For example, the design phase of industrial products related to Internet of Things (IoT) systems could take advantage of the power of DEVS design pattern-based approach. DEVS gives a high level of abstraction to easily specify temporal constraints within intelligent hybrid cyberphysical system design that are difficult to manage otherwise [41].

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation: Discrete Event Iterative System Computational Foundations*; Academic Press: New York, NY, USA, 2018.
2. Zaft, G.C.; Zeigler, B.P. Discrete Event Simulation and Social Science: The XeriScape Artificial Society. In *SCI 2002*; SCS Publications: Orlando, FL, USA, 2002.
3. Zeigler, B.P.; Muzy, A. Temporal Modeling of Neural Net Input/Output Behaviors: The Case of XOR. *Systems* **2017**, *5*, 7. [CrossRef]
4. Gourlis, G.; Kovacic, I. Energy efficient operation of industrial facilities: The role of the building in simulation-based optimization. *IOP Conf. Ser. Earth Environ. Sci.* **2020**, *410*, 012019. [CrossRef]
5. Mittal, S.; Risco-Martin, J.L. DEVSML Studio: A framework for integrating domain-specific languages for discrete and continuous hybrid systems into DEVS-Based M&S environment. In Proceedings of the Summer Computer Simulation Conference, Montreal, QC, Canada, 24–27 July 2016.
6. Mittal, S.; Tolk, A. *Complexity Challenges in Cyber Physical Systems*; Wiley: New York, NY, USA, 2018.
7. Mittal, S.; Zeigler, B.P. The practice of modeling and simulation in cyber environments. In *The Profession of Modeling and Simulation*; Tolk, A., Oren, T., Eds.; Wiley: Hoboken, NJ, USA, 2017; pp. 223–264.
8. Vangheluwe, H. DEVS as a common denominator for multi-formalism hybrid systems modelling. In Proceedings of the IEEE International Symposium on Computer-Aided Control System Design, Anchorage, AK, USA, 25–27 September 2000; pp. 129–134.
9. Zeigler, B.; Kim, D. *Multi-Resolution Modeling for Adaptive UAV Service Systems*; ANSS: Canonsburg, PE, USA, 2019.
10. Zeigler, B.P. Hybrid Iterative System Specification of Cyberphysical Systems: Neurocognitive Behavior Application. In Proceedings of the 2020 Spring Simulation Conference (SpringSim), Virtual Conference, 18–21 May 2020.
11. Zeigler, B.P. Discrete Event Abstraction: An Emerging Paradigm for Modeling Complex Adaptive Systems. In *Perspectives on Adaptation in Natural and Artificial Systems*; Oxford University Press: Oxford, UK, 2004.
12. Gigerenzer, G.; Goldstein, D.G. Reasoning the Fast and Frugal Way: Models of Bounded Rationality. In *Judgment and Decision Making*; Connolly, T., Ed.; Cambridge University Press: Cambridge, UK, 2000; pp. 621–650.
13. Gigerenzer, G.; Todd, P.M. *Simple Heuristics That Make Us Smart*; Oxford University Press: Oxford, UK, 1999.
14. Tóth, M.; Chytilek, R. Fast, frugal and correct? An experimental study on the influence of time scarcity and quantity of information on the voter decision making process. *Public Choice* **2018**, *177*, 67–86. [CrossRef]
15. Luh, C.; Zeigler, B.P. Abstracting Event-Based Control Models for High Autonomy Systems. *IEEE Trans. Syst. Man Cybern.* **1993**, *23*, 42–54. [CrossRef]
16. Muzy, A.; Zeigler, B.P.; Grammont, F. Iterative Specification as a Modeling and Simulation Formalism for I/O General Systems. *IEEE Syst. J.* **2017**, *12*, 2982–2993. [CrossRef]
17. Zeigler, B.P. *DEVS Representation of Dynamical Systems: Event-based Intelligent Control*; IEEE: Piscataway, NJ, USA, 1989; Volume 77, pp. 72–80.
18. Pinker, S. *How the Mind Works*; W.W. Norton: New York, NY, USA, 1997.

19. Seck, M.; Frydman, C.F.; Giambiasi, N. Using DEVS for Modeling and Simulation of Human Behavior. In *International Conference on AI, Simulation, and Planning in High Autonomy Systems*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 692–698.
20. Kleene, S.C. *Representation of Events in Nerve Nets and Finite Automata*; Automata Studies; Princeton University Press: Princeton, NJ, USA, 1956.
21. Maas, W.; Bishop, C.M. *Pulsed Neural Networks*; MIT Press: Cambridge, MA, USA, 1999; p. 377.
22. Watts, L. Event-driven simulation of networks of spiking neurons. In *Advances in Neural Information Processing Systems*; Morgan Kaufmann: San Mateo, CA, USA, 1994; pp. 927–934.
23. Delorme, A.; Gautrais, J.; van Rullen, R.; Thorpe, S. SpikeNET: A simulator for modeling large networks of integrate and fire neurons. In *Computational Neuronscience: Trends in Research*; Elsevier Science: Amsterdam, The Netherlands, 1999; pp. 989–996.
24. Mascart, C.; Scarella, G.; Reynaud-Bouretb, P.; Muzy, A. Simulation scalability of large brain neuronal networks thanks to time asynchrony. *Biorxiv* **2021**. [CrossRef]
25. Adegbija, T. DINGO: A Modeling Approach for Translating Spiking Neural Networks to Hardware Accelerators. In Proceedings of the Design Automation Conference, San Francisco, CA, USA, 10–14 July 2022; ACM: New York, NY, USA, in process.
26. Seo, C.; Zeigler, B.; Coop, R.; Kim, D. DEVS Modeling and Simulation Methodology with MS4Me Software TMS. In Proceedings of the 2013 Spring Simulation Multiconference, San Diego, CA, USA, 7–10 April 2013.
27. Zeigler, B.P.; Sarjoughian, H. *Modeling and Simulation of Systems of Systems*; Springer Pub. Co: New York, NY, USA, 2017.
28. Castro, S.; Mosterman, P.J.; Rajhans, A.H.; Valenti, R.G. Challenges in the Operation and Design of Intelligent Cyber-Physical Systems. In *Complexity Challenges in Cyber Physical Systems: Using Modeling and Simulation (M&S) to Support Intelligence, Adaptation and Autonomy*; John and Wiley and Sons: Hoboken, NJ, USA, 2019.
29. Wach, P.; Zeigler, B.P.; Salado, A. Conjoining Wymore's Systems Theoretic Framework and the DEVS Modeling Formalism: Toward Scientific Foundations for MBSE. *Appl. Sci.* **2021**, *11*, 4936. [CrossRef]
30. Krohn, K.; Rhodes, J. Algebraic theory of machines, I. *Trans. Am. Math. Soc.* **1965**, *116*, 450–464. [CrossRef]
31. Vahie, S. Dynamic Neuronal Ensembles: Neurobiologically Inspired Discrete Event Neural Networks. In *Discrete Event Modeling and Simulation Technologies*; Sarjoughian, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2000.
32. Vahie, S.; Jouppi, N. Dynamic Neuronal Ensembles: A New Paradigm for Learning & Control. In *AI, Simulation and Planning in High Autonomy Systems*; SCS Publications: San Diego, CA, USA, 1996.
33. Gautrais, J.; Thorpe, S. Rate Coding Versus Temporal Coding: A Theoretical Approach. *BioSystems* **1998**, *48*, 57–65. [CrossRef]
34. Van Rullen, R.; Gautrais, J.; Delorme, A.; Thorpe, S. Face Processing Using One Spike Per Neurone. *BioSystems* **1998**, *48*, 229–239. [CrossRef]
35. McNaughton, R.; Yamada, H. Regular Expressions and State Graphs for Automata. In *IRE Transactions on Electronic Computers*; IEEE: Piscataway, NJ, USA, 1960; Volume EC-9.
36. Yu, S. Regular Languages. In *Handbook of Formal Languages*; Rozenberg, G., Salomaa, A., Eds.; Springer: Berlin/Heidelberg, Germany, 1997.
37. McCullogh, W.S.; Pitts, W.H. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
38. White, N. Multiple Memory Systems. Available online: http://www.scholarpedia.org/article/Multiple_memory_systems (accessed on 14 December 2021).
39. Feinerman, O.; Rotem, A.; Moses, E. Reliable neuronal logic devices from patterned hippocampal cultures. *Nat. Phys.* **2008**, *4*, 967–973. [CrossRef]
40. Petersen, S.E.; Sporns, O. Networks and Cognitive Architectures. *Cell Rev.* **2015**, *88*, 1–236.
41. Capocchi, L.; Santucci, J.-F.; Tigli, J.-Y.; Gonnin, T.; Lavirotte, S.; Rocher, G. A new discrete-event simulation based approach for validating actuation conflict management in IoT systems. In Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM), Fairfax, VA, USA, 19–22 July 2021.