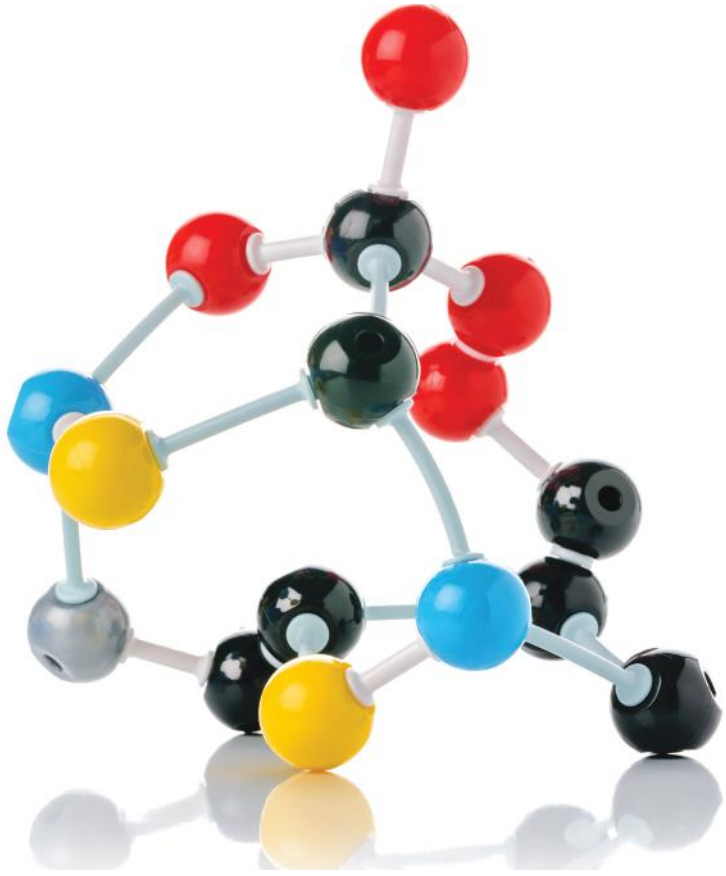
Editors: Barry I. Schneider, bis@nist.gov | Gabriel A. Wainer, gwainer@sce.carleton.ca



Designing Biological Simulation Models Using Formalism-Based Functional and Spatial Decompositions

Rhys Goldstein | Autodesk Research
Gabriel A. Wainer | Carleton University

Simulation is becoming an increasingly common tool among biologists, complementing traditional experimental techniques. As Hiroaki Kitano explained,[1] experimental data is first used to form a hypothesis, which can then be investigated with a simulation. Predictions made by the simulation can, in turn, be tested using in vitro and in vivo studies, with new experimental data leading to new hypotheses. This iterative process can be applied to basic research on biological systems, as well as in the development of drugs and other treatments.

The simulation of biological systems poses many technical challenges. Among these are accurate model parameter selection, model validation, and code optimization for computational efficiency. Our interests lie in the development of well-designed simulation software, which is difficult for two reasons: first, the systems (and corresponding models) are often complex, and second, realistic simulations might require the integration of multiple complex algorithms (such as simulating the deformation of a cell membrane surrounded by reacting and diffusing chemicals in a changing electric field).

In most engineering applications, we can address a large system's complexity by partitioning it into simpler subsystems. Herbert Sauro and his colleagues[2] suggested

 1521-9615/15/$31.00 © 2015 IEEE Copublished by the IEEE CS and the AIP November/December 2015

that complex biological systems be conceptually modularized in an analogous manner, and they recommend that modeling formalisms be adopted to support this approach. The Discrete Event System Specification (DEVS) is one such formalism.[3] Using DEVS, a simulation program is partitioned into a simulator and a model—we can address the model's complexity by subdividing it into simpler submodels, and those submodels can in turn be subdivided in a hierarchical fashion.

Having selected a modeling formalism (DEVS) and a means of addressing complexity (hierarchical model design), an important question remains: How should the system of interest be decomposed? Here, we focus on three options: topological decomposition, functional decomposition, and spatial decomposition.

Topological decomposition tries to define separate submodels for separate biological entities and then link the submodels in a way that resembles real-world connections. This is the approach that Roland Ewald and colleagues adopted,[4] using separate DEVS submodels to represent a cell membrane, a cytoplasm, and a nucleus, connected in sequence. They identified a weakness of this approach: the difficulty of representing interactions involving three or more entities. We argue that drawbacks encountered using a particular formalism must be considered in the context of the chosen decomposition strategy. It isn't clear whether interactions among three or more entities are difficult to model using DEVS if the hierarchy isn't based on topology.

Functional decomposition involves the separation of different aspects of a real-world system. These aspects can be functions of biological systems (such as the transfer of information along a nerve cell in the form of an action potential) or physical processes (such as diffusion inside a cell membrane). Given a hierarchical model based on a functional decomposition, a submodel is unlikely to represent a single entity; rather, it might represent a single effect for a large number of entities.

Spatial decomposition partitions the space into discrete regions represented by model instances. Such decompositions typically take the form of a 2D grid or 3D rectangular lattice. This technique is supported by the Cell-DEVS formalism.[5]

In our research, which involves the self-assembly and deformation of biological structures in 3D, we find it useful to partition spatial regions as described above but only at lower levels in a DEVS model hierarchy. At upper levels, we define submodels for different simulation algorithms representing aspects of the system. In other words, we adopt a function decomposition near the top of the hierarchy and spatial decomposition near the bottom. To demonstrate this approach, we present a hierarchical DEVS model of a presynaptic nerve terminal, the compartment in a nerve cell at which an action potential can excite an adjacent nerve cell.

Discrete Event System Specification

In the mid-1970s, with the conviction that a novel theory was needed for discrete event simulation, Bernard Zeigler invented DEVS, a general modeling formalism that's essentially a set of conventions for the formal description of a wide range of systems. A DEVS model is designed by defining certain mathematical sets and functions, collectively called an *atomic model*. Coupled models are composed of submodels that are themselves either atomic or coupled. The nesting of coupled models within coupled models is the mechanism by which DEVS supports hierarchical model design.

The formalism emphasizes the separation between simulators and models. A DEVS simulator is model-independent in the sense that it should carry out a simulation for any well-defined DEVS model, regardless of what the model represents.

Despite the popularity of the DEVS formalism and the widespread use of simulation in the study of biological systems, the application of DEVS to biological models is practiced by relatively few researchers. A group led by Adelinde Uhrmacher at the University of Rostock applied DEVS to various nonspatial models of biological systems. Ewald's team[4] compared DEVS to stochastic Petri nets, stochastic π-calculus, and StateCharts, using a DEVS model hierarchy designed by coupling submodels that represent various entities of a biological system: a cell's membrane, nucleus, and cytoplasm. The researchers simulated interactions between such entities by passing messages between the DEVS models. Because messages are passed independently of one another, the researchers claim that DEVS is unsuitable for modeling interactions that involve three or more entities. These claims might indicate considering alternatives to DEVS, but we explored alternatives to topological decomposition.

Numerous variants of DEVS have been developed. Cell-DEVS,[5] for instance, has been used to design cellular biological models, representing real-world systems as cell spaces in which each cell is defined as a DEVS model. This technique facilitates the specification of discrete-event cell spaces

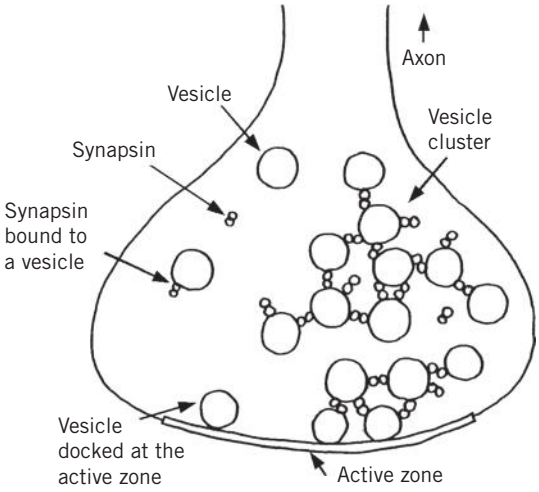


Figure 1. Illustration of a presynaptic nerve terminal (PNT). Inside a PNT are tens to hundreds of neurotransmitter-containing synaptic vesicles, some of which are docked to a region of the membrane called the active zone.

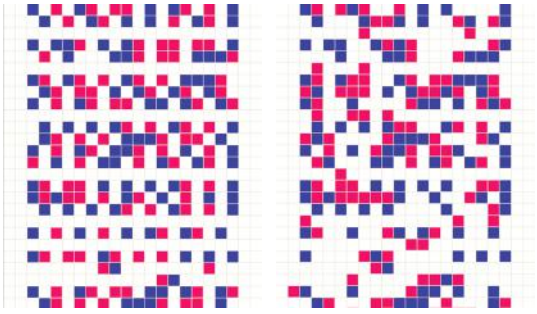


Figure 2. Results showing vesicles (blue) and synapsin (red) in their initial (left) and final (right) configurations. This rough approximation fails to capture the membrane enclosing a presynaptic compartment.

and improves their definition by making the timing specification more expressive. Cell-DEVS has also been used recently to implement computational fluid dynamics (CFD),[6] which has several applications in large-scale biological models.

Presynaptic Nerve Terminal Models

To demonstrate our approach to hierarchical DEVS model design, we focus on the simulation of a compartment called the *presynaptic nerve terminal* (PNT), depicted in Figure 1. Numerous PNTs can be found on the ends of a single neuron. Inside a PNT are tens to hundreds of neurotransmitter-containing synaptic vesicles,[7] some of which are docked to a region of the membrane called the *active zone*. When an action potential arrives from the nerve cell's axon, certain docked vesicles can release their neurotransmitters outside the compartment, which could provoke another action potential in an adjacent nerve cell. When a docked vesicle releases its neurotransmitters in this fashion, it can undergo a process called *exocytosis*, in which it fuses with the nerve cell membrane. In addition, PNTs contain hundreds of protein called *synapsin*,[8] which bind with vesicles to form clusters. An action potential triggers chemical reactions that cause synapsins to lose their affinity for vesicles.

Experimental results[9] suggest that synapsins help maintain several vesicles in the active zone's vicinity, which intuitively should increase the chance that a sequence of action potentials is transmitted from one neuron to the next. One rationale for modeling a PNT is to quantify the relationship between synapsin concentration and the availability of docked vesicles, which could aid in investigating this theory.[10] Population-based methods[11] that track concentrations of particles are inadequate for this purpose because they don't track the locations of individual particles.

Discrete-Space PNT Models

An early effort to model vesicle clusters used Cell-DEVS to predict the number of vesicles released from the reserve pool as a function of time under the influence of action potentials at differing frequencies.[12] The molecular interactions of synapsin and vesicles were modeled as they occur inside a nerve cell, with the model describing the behavior of synapsin movements until they reached a vesicle and bonded to it based on a given onrate. Once synapsins become bonded with vesicles, an offrate is used to model the breaking of bonds. Figure 2a shows a grid at the initial point, and Figure 2b shows how bonds were formed and how the corresponding cells change their values to represent the binding.

Although the cellular model in Figure 2 is simple to modify and describe, it's a very rough approximation that, among other limitations, fails to capture the membrane enclosing a presynaptic compartment. Therefore, we explored enhanced models based on cellular models,[13] as Figure 3 shows.

A submodel's state can represent a vesicle, a synapsin, an empty space inside the compartment, a part of the membrane, or a part of the active zone at the bottom of the membrane. The figure shows three snapshots: the left image shows clusters—here, the first action potential arrived immediately,

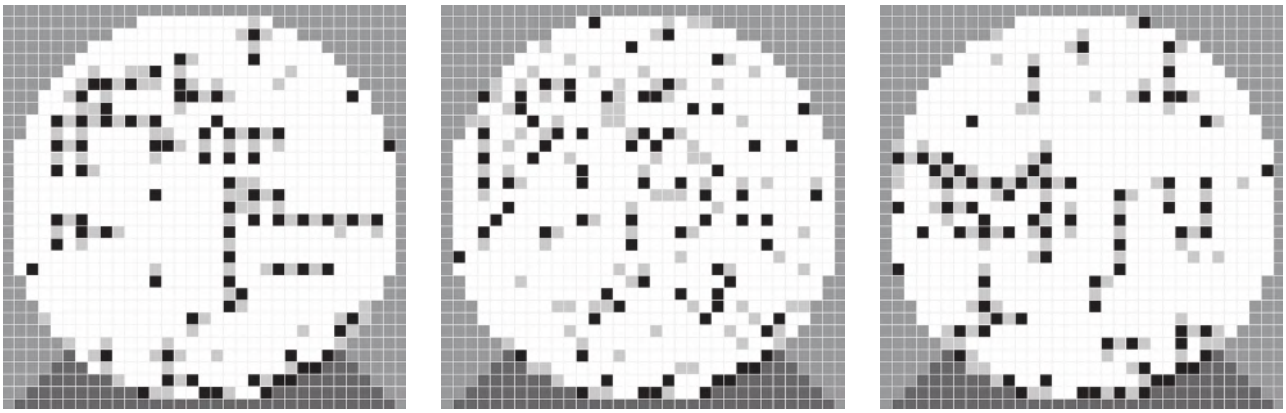


Figure 3. A cellular model of a PNT. Vesicles (black cells) form clusters with synapsin (light gray cells) inside a circular compartment. The left image shows clusters—here, the first action potential arrived immediately, after which the vesicles and synapsins in the large clusters dispersed (center image). The clusters later reformed (right image).

after which the vesicles and synapsins in the large clusters dispersed (center image). The clusters later reformed (right image).

These models use a hierarchical decomposition that partitions the space into rectangular cells and considers both how they're occupied (synapsin, vesicles) and their bindings, which influenced the continuous-space PNT models.

Continuous-Space PNT Models

The difference between a discrete-space model and a continuous-space one is that, in the latter case, spatial coordinates aren't restricted to discrete points on a lattice. With a continuous-space PNT model, vesicles can be represented as spherical particles that can move in any direction. This approach adds complexity to the model but allows for more realistic simulations.

Our continuous-space algorithm, called the *tethered particle system* (TPS) was invented for the simulation of self-assembling deformable biological structures.[14] In TPS, numerous particles of different sizes are tracked as they move through space. As the simulation progresses, collisions between pairs of particles are continually detected. When a collision occurs, an impulse is calculated that alters the trajectories of the particles involved. The characteristic that differentiates TPS from similar impulse-based methods such as billiard simulations is the fact that certain pairs of particles can become tethered together. A collision between separating tethered particles can cause them to retract inward instead of rebounding outward.

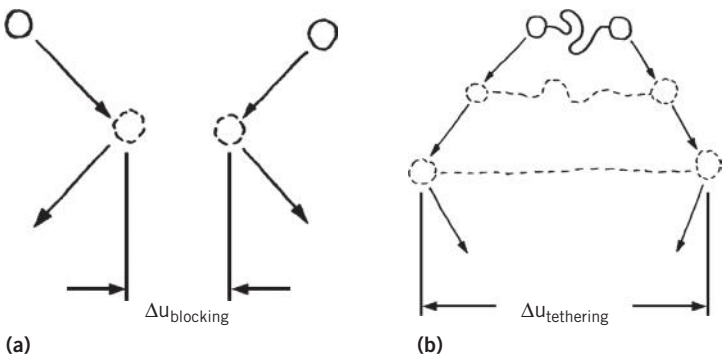


Figure 4. Two types of collisions in tethered particle system (TPS) simulation: (a) blocking collision and (b) tethering collision. A blocking collision occurs when two approaching particles reach an inner limiting distance $\Delta u_{blocking}$, causing the particles to rebound outward. If two separating tethered particles reach an outer limiting distance $\Delta u_{tethering}$, and if the particles remain tethered, then a tethering collision causes the particles to retract.

The distances between certain pairs of particles are thus constrained, and these constraints allow a variety of deformable structures to be represented. TPS provides a relatively simple way to allow deformable biological structures to assemble themselves from rigid particles representing proteins and other biological entities. We can also represent the Brownian motion exhibited by these small biological objects by applying random impulses, randomized changes in momentum applied at randomized times.

Figure 4 illustrates the two types of collisions in a TPS: blocking and tethering. A blocking

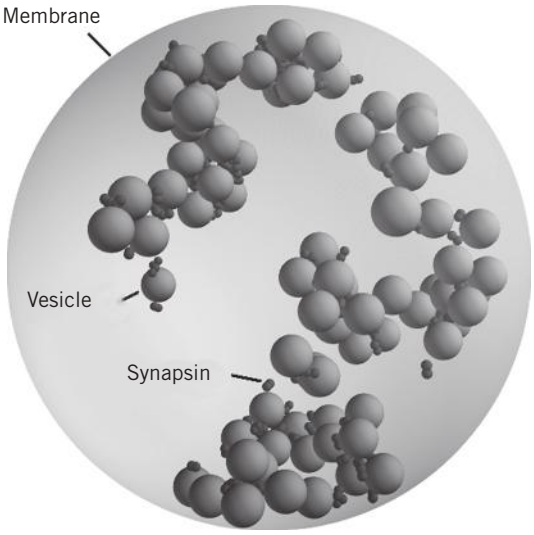


Figure 5. A continuous-space model of a PNT. Here, vesicles (large spheres) form clusters with synapsins inside a compartment.

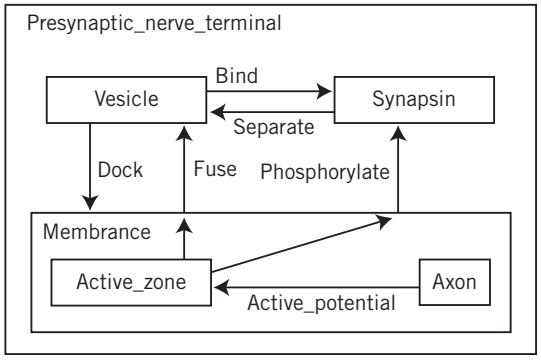


Figure 6. A hypothetical PNT model featuring a hierarchy based on topology. The outer presynaptic_nerve_terminal model is divided into a vesicle, a synapsin, and a membrane submodel. The membrane is in turn divided into an active_zone and an axon model. Various links between the submodels are added based on interactions between the topologically related biological entities.

collision occurs when two approaching particles reach an inner limiting distance $\Delta u_{blocking}$, as in Figure 4a. This causes the particles to rebound outward. If two separating tethered particles reach an outer limiting distance $\Delta u_{tethering}$, and if the particles remain tethered, then a tethering collision causes the particles to retract. The unraveling cord in Figure 4b illustrates this effect, although the cord isn't explicitly modeled.

It might seem counterintuitive at first to use TPS to model deformable structures. With a deformable structure, we would expect any sort of inward motion to decelerate gradually as the object is compressed. With TPS, however, two approaching particles will continue to move at a constant speed until the blocking distance is reached. There's no virtual spring to slow down the particles gradually. Only when we have a large number of tethered particles does deformation-like behavior occur: the tethered particles are compressed, and the number of internal collisions increases, which in turn slows the inward motion. The impulses applied randomly to individual particles to simulate Brownian motion ensure that the overall system becomes realistically chaotic, even if the particles are initially motionless in a contrived configuration.

TPS differs from most alternative methods. First, it adheres to a discrete-event simulation approach instead of a discrete-time approach; there's no fixed time step. To implement TPS, we must predict the time associated with all anticipated future collisions between particles. When a collision occurs, time advances to the most imminent future collision time, resulting in a nonuniform sequence of time steps.

Another key difference between TPS and related methods is the way particle interactions are handled. For instance, in the Cellular Dynamic Simulator,[15] two colliding particles are often replaced with a single product particle. Although TPS could be extended to accommodate this technique, such reactions are generally represented by tethering the two colliding particles. The particles are still treated independently, however—the fact that they behave as part of the same structure emerges as a result of subsequent tethering collisions. This effect is best observed in animations of TPS simulations, where deformable structures (such as the vesicle-synapsin clusters in Figure 5)[16] self-assemble from individual particles.

The method most closely related to TPS is discrete-molecular dynamics (DMD).[17] Unlike traditional molecular dynamics, DMD treats potential fields surrounding atoms as step functions, giving rise to constant particle velocities that are characteristic of TPS. Note that TPS targets much larger structures, accommodates greater discrepancies in particle sizes, and is motivated by the need to enforce constraints rather than a desire to quantify electrostatic potentials.

DEVS-Based Continuous-Space PNT Model

Topological decomposition, devoting separate submodels to separate entities, is an intuitive way to design a hierarchical DEVS model of a PNT. Figure 6 illustrates how such a model might be structured. The outer *presynaptic_nerve_terminal*

model is divided into a vesicle, a synapsin, and a membrane submodel. The membrane is in turn divided into an *active_zone* and an axon model. Various links between the submodels are added based on interactions between the topologically related biological entities; the binding of vesicles and synapsins, and the propagation of an action potential, are examples.

Although intuitive, topological decomposition has several drawbacks for certain models, in particular, the continuous-space presynaptic nerve terminal model based on the TPS method. In TPS, an individual particle has little behavior on its own. Almost all the complexity pertains to the detection and resolution of collisions, each of which involves multiple particles. In Figure 6, it's unclear where collision detection and resolution code should be located. Further issues arise if particles must be added or removed from the system during a simulation.

We adopt functional decomposition at upper levels of our model hierarchy. This approach lets us track the state of multiple biological entities in a single DEVS model, thus we can have interactions between three or more entities. In addition, if we choose to separate functions or physical processes at the upper levels, it's still possible to partition space at lower levels—this is how we designed a continuous-space DEVS model of a PNT.

TPS has three distinct aspects: the generation of random impulses applied to individual particles, the detection of collisions between particles, and the calculation of new particle trajectories in response to impulses or collisions. Accordingly, we designed a TPS DEVS model with three submodels: *RI*, which generates random impulses; *detector*, which detects collisions; and *responder*, which calculates new trajectories. Note that this hierarchy is an example of functional decomposition.

The key interaction of TPS model is represented by the loop at the bottom right of Figure 7. When two particles collide, the detector sends a collision message to the responder, which outputs one response message for each affected particle. These messages describe the particle's new trajectory, sent to the detector so that future collision times can be recalculated. The loading and restitution output messages allow three or more particles to be involved in a collision.[14]

Other interactions include impulses that originate from either the RI submodel or a TPS input and alter a particle's trajectory. These messages enter the responder and result in an impulse output

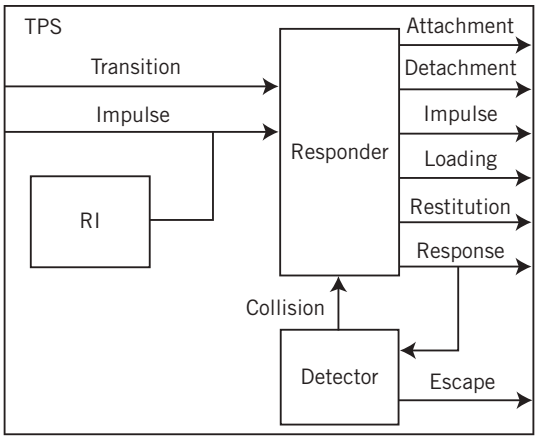


Figure 7. Structure of TPS DEVS model. When two particles collide, the detector sends a collision message to the responder, which outputs one response message for each affected particle. These messages describe the particle's new trajectory, sent to the detector so that future collision times can be recalculated.

message and one or more response messages. Particle collisions can cause two separate particles to become "tethered," in which case the responder outputs an attachment message, or they can cause two tethered particles to separate, in which case a detachment message is output. Input transition messages can affect the nature in which particles attach or detach. Finally, the detector can output an escape message if a particle strays too far from the center of the region represented by the model.

We preserve the principle of encapsulation, as each submodel has its own representation of the entire system. In general, some information will be common to multiple submodel-specific representations, while some information will be specific to certain representations. For example, each particle's mass is relevant to responder and RI, but not to detector—likewise, each particle's position and velocity are relevant to responder and detector, but not to RI. If a particle's position or velocity changes in the responder, it's communicated to the detector via a message, which preserves encapsulation.

An Atomic Model Example

The DEVS RI model generates random impulses for any particle in TPS. Because the model is separated from the collision detection and response models, its definition is compact and self-contained.

DEVS atomic models in general take the form $<X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta>$. At any given time,

a DEVS model has a state $s \in S$. After an event, the simulator evaluates the time advance function *ta* to schedule the internal event. Should this time elapse, the output function λ is invoked to obtain an output value $y \in Y$, and the internal transition function δ_{int} yields a new state. If an input $x \in X$ is received before *ta* elapses, the simulator applies the external transition function δ_{ext} instead to obtain the new state.

In our case, RI has no inputs, thus $X = \varnothing$ and δ_{ext} is never invoked. The set of output values Y includes all vectors of the form [*id*, Δp], where *id* identifies a particle and Δp is a randomly generated impulse to be applied to that particle. The set of states S includes all vectors of the form [*t*, *FEL*], where the state variable *t* is the time of the last transition, and *FEL* is a list of future events. The output function λ uses these state variables to construct an output value:

$y = \lambda(s)$
1. $[t, FEL] = s$ Extracts the state variables
2. $[id, t'] = next\_event(FEL)$ Identifies the next random impulse
3. $\Delta p = impulse(id)$ Generates the impulse value
4. $y = [id, \Delta p]$ Constructs the output value

Once the value $\lambda(s)$ is output, a new state is obtained from the internal transition function δ_{int}:

$y = \delta_{\text{int}}(s)$
1. $[t, FEL] = s$ Extracts the state variables
2. $[id, t'] = next\_event(FEL)$ Identifies the next random impulse
3. $t_{FEL} = t' + detect(id)$ Calculates a new impulse time
4. $FEL' = add\_event(FEL, id, t_{FEL})$ Updates the future events list
5. $s' = [t', FEL']$ Constructs the new state

After the state of RI is updated, the time remaining before the next transition is obtained from the time advance function *ta*:

$\Delta t = ta(s)$
1. $[t, FEL] = s$ Extracts the state variables
2. $[id, t'] = next\_event(FEL)$ Identifies the next random impulse
3. $\Delta t = t' - t$ Calculates the remaining time

Once an atomic model is specified using DEVS conventions, it can be implemented using a DEVS-based tool or library. The code for the RI model, developed for a DEVS library written in Python, is reproduced in Figure 8.

The four different DEVS functions are defined in the main function as *RI_DEVS*, and the four implemented functions are returned in the structures right at the end, such as *RI* = [*delta_ext, delta_int, ta*] and [*init_RI, RI*]. In the selected Python library, the output function λ is absorbed into the internal transition function *delta_int*. The *init_RI* function initializes the model.

Spatial Decomposition for Collision Detection

As mentioned earlier, even if upper levels of a DEVS model hierarchy are dedicated to the separation of biological functions, it's still possible to partition space at lower levels. One advantage is that different spatial decompositions can be adopted for different algorithms. In our case, only the detector can benefit from a spatial decomposition. The RI and responder submodels are therefore defined as atomic models, whereas the detector is a coupled model.

Within detector are a set of DEVS submodels representing different subvolumes of space. Collisions are detected independently in each of these submodels for the associated subvolume, which is more computationally efficient than detecting collisions for the entire space. Figure 9 shows this decomposition of space.

Although collisions are detected within cubic subvolumes, we need to track which particles are close to a subvolume to include in its collision detection. It turns out that it's somewhat cumbersome to check whether a spherical particle is in a cubic subvolume but extremely easy to determine if it's in a spherical region. For this reason, each subvolume features two concentric encompassing spheres. The inner sphere detects incoming particles for the collision detection calculations. The outer sphere detects outgoing particles that can be excluded from the process.

Figures 10 and 11 illustrate the tracking of particles in 2D. In Figure 10, particle A is exiting the outer circle around subvolume [0, 2]. Accordingly, it becomes excluded from this subvolume's search for future collisions. In Figure 11, particles A and B are both within the inner circles of subvolumes [0, 0] and [1, 0]. Because the collision actually occurs in [1, 0], this is the only subvolume that schedules the collision.

The TPS model presented earlier can be used as the uppermost-level DEVS, but it can also be incorporated into a larger DEVS hierarchy. When

```
def RI_DEVS (n_dim, Omega_psi):
    empty_FEL = future_events_list["empty_FEL"]
    delta_FEL = future_events_list["delta_FEL"]
    event_FEL = future_events_list["event_FEL"]
    pr_None = DEVS["pr_None"]
    detect_RI = tethered_particle_system["detect_RI"]
    impulse_RI = tethered_particle_system["impulse_RI"]

    def init_RI (Psi):
        t = 0
        SPC = {}
        FEL = empty_FEL()
        for id_A in Psi.keys():
            SPC[id_A] = Psi[id_A]"spc"][
            spc_A = SPC[id_A]
            t_A = t + detect_RI(spc_A, Omega_psi)
            FEL = delta_FEL(FEL, id_A, t_A, pr_None)
        s = [t, SPC, FEL]
        return s

    def delta_ext (s, Delta_t_el, x):
        raise ValueError("invalid input for 'RI', accepts no inputs")

    def delta_int(s):
        [t, SPC, FEL] = s
        [id_A, t_] = event_FEL(FEL)
        spc_A = SPC[id_A]
        t_A = t_ + detect_RI(spc_A, Omega_psi)
        FEL_ = delta_FEL(FEL, id_A, t_A, pr_None)
        s_ = [t_, SPC, FEL_]
        Delta_p = impulse_RI(n_dim, spc_A, Omega_psi)
        Y = [["impulse", [id_A, Delta_p]]]
        return [s_, Y]

    def ta(s):
        [t, SPC, FEL] = s
        [id_A, t_A] = event_FEL(FEL)
        Delta_t_int = max([0.0, t_A -t])
        return Delta_t_int
    RI = [delta_ext, delta_int, ta]
    return [init_RI, RI]
```

Figure 8. Python code for the DEVS Random Impulse (RI) model.

applying TPS to a PNT, our simulation was complicated by action potentials and the fusing of vesicles with the membrane of the presynaptic compartment. We therefore embedded the TPS model in a coupled model named PNT, including *models action*, which triggers action potentials, and *fusion*, which determines when vesicles fuse with the nerve cell membrane. Figure 12 shows the structure of the PNT model (it's another example of functional decomposition).

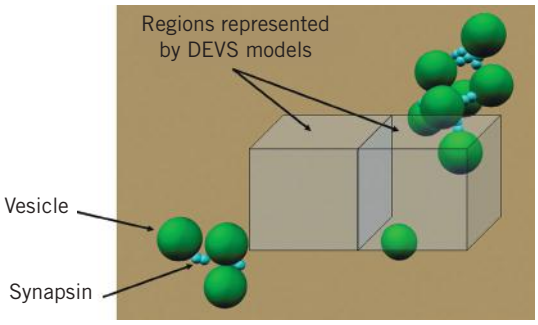


Figure 9. The spatial decomposition used to detect collisions in the DEVS-based implementation of the TPS algorithm.

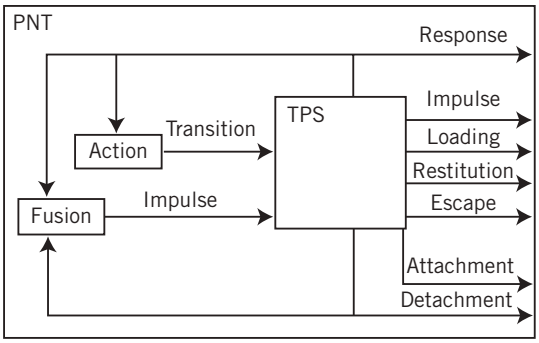


Figure 12. The structure of the PNT DEVS model. Because links between DEVS submodels are associated with a coupled model, the submodels themselves can interoperate without explicitly referencing one another.

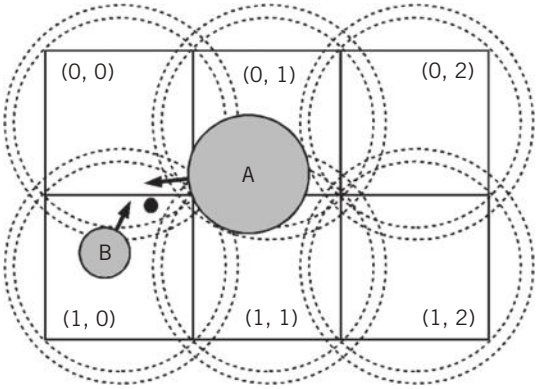


Figure 10. A scenario in which two particles approach one another. Particle A is exiting the outer circle around subvolume [0, 2]. Accordingly, it becomes excluded from this subvolume's search for future collisions.

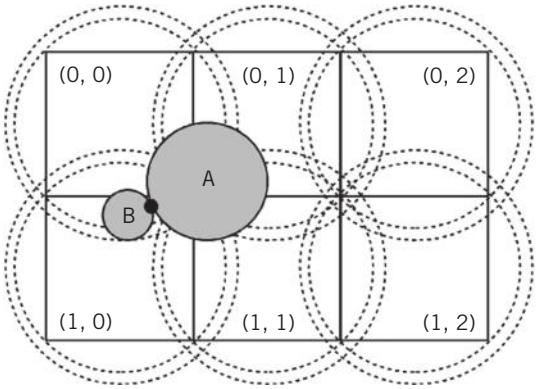


Figure 11. The same scenario as in Figure 10, but at the later time when the two particles collide. A and B are both within the inner circles of subvolumes [0, 0] and [1, 0]. Because the collision actually occurs in [1, 0], this is the only subvolume that schedules the collision.

Because links between DEVS submodels are associated with a coupled model, the submodels themselves can interoperate without explicitly referencing one another. For example, although the action model interacts with the TPS model, its definition wouldn't mention the identifier "TPS." This convention distinguishes DEVS-based hierarchies from typical object-oriented networks in which interacting objects tend to invoke each other's methods explicitly.

Simulating Nerve Terminals

We've executed numerous simulations in collaboration with a research team led by James Cheetham in the Department of Biology at Carleton University. His team has explored a variety of simulation scenarios of interest, the details of which appear elsewhere.[13,14,16,18] Animations of these results appear online at www.youtube.com/arslab.

Figure 13 shows one of these simulation results based on the model described earlier. A cluster of vesicles, bound to one another by synapsin, has formed in the vicinity of the active zone where the neurotransmitters are ultimately released. These behaviors aren't programmed explicitly but instead emerge from the execution of the TPS rules.

Figure 14 shows a simulation in which vesicles in a PNT respond to an action potential. This is initiated at time 600, after all vesicles have become bound in a single cluster at the active zone (Figure 14a). After 20 time units, a vesicle can be seen escaping the compartment in a simulated exocytosis (Figure 14b). At time 680, recently freed synapsin can be seen above the visibly disrupted cluster (Figure 14c). Many of the freed synapsin eventually bind with vesicles again, and the cluster reforms (Figure 14d).

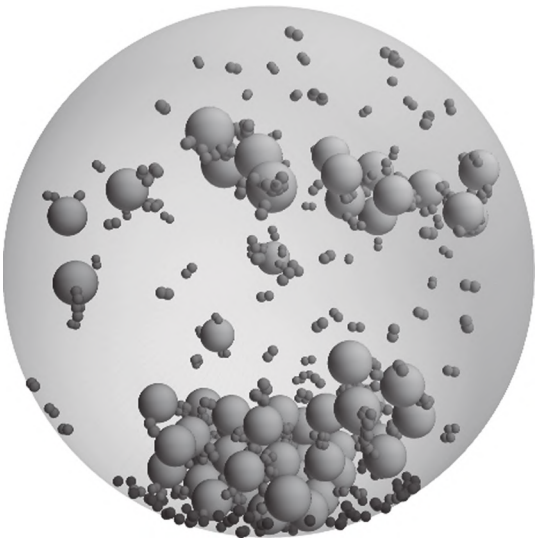


Figure 13. Simulation snapshot of a TPS model of a PNT.

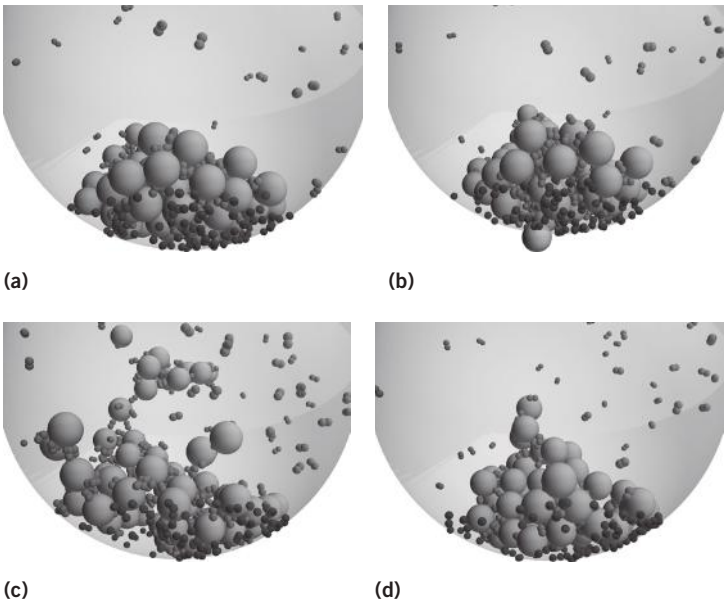


Figure 14. Four snapshots of a vesicle cluster reacting to an action potential in a simulated presynaptic nerve terminal: (a) [$t = 600$], (b) [$t = 620$], (c) [$t = 680$], and (d) [$t = 900$].

Figure 15 shows several action potentials affecting a number of vesicles in the PNT (which decreases with each action potential) and the average normalized distance of vesicles from the active zone. The distance generally decreases as vesicles cluster at the active zone. When action potentials occur, vesicles released from the cluster cause a temporary increase in this distance.

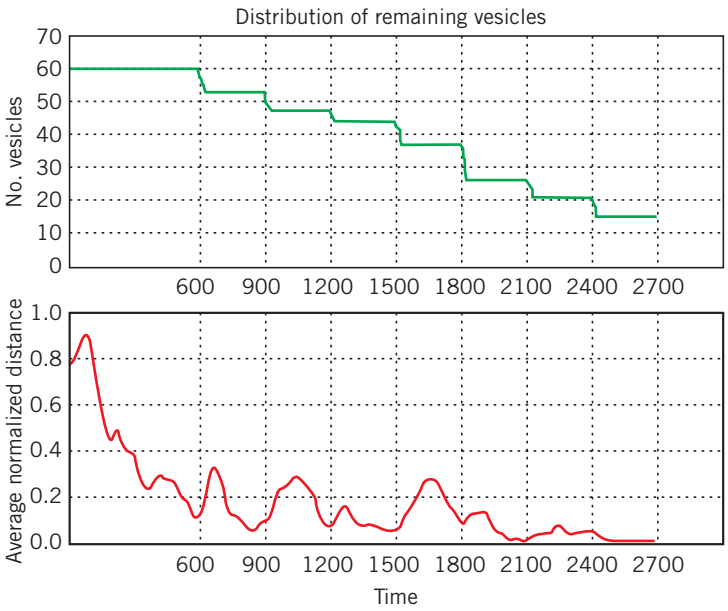


Figure 15. Simulated depletion of vesicles and their average distance from the active zone, in response to action potentials.

Simulations like the ones presented here help biologists investigate the role of synapsin in the human brain. Experimental results suggest that synapsin helps maintain several vesicles in the vicinity of the active zone, which in turn increases the chance that a sequence of action potentials will be transmitted from one neuron to the next. Simulations can be used to quantify the relationship between synapsin concentration and the availability of docked vesicles. An iterative research process, involving both simulation and experimentation, could lead to a better understanding of neurotransmission coupled with an ability to predict behavior inside presynaptic compartments.

By applying DEVS to TPS, we designed the first continuous-space DEVS models intended for the simulation of biological systems. We adopted an alternative formalism to address the shortcomings of previous approaches. We used a spatial decomposition at a lower level in our hierarchy; at upper levels, our hierarchy exhibits a functional decomposition that separates various real-world effects. This approach can be useful for the design of biological systems: separate different functions or algorithms at upper levels in a model hierarchy, partition space at lower levels, and refrain from associating particular models with specific biological entities.

In the near future, systems biologists will likely wish to simulate the dynamics of deformable structures in combination with the reaction and diffusion of chemicals, and perhaps the propagation of electric fields as well. The DEVS formalism provides a means to achieve this integration. Separate DEVS models can be defined for different simulation algorithms—by linking these models together, the algorithms themselves can be combined. ■

References

1. H. Kitano, "Computational Systems Biology," *Nature*, vol. 420, 2002, pp. 206–210.
2. H.M. Sauro et al., "Challenges for Modeling and Simulation Methods in Systems Biology," *Proc. Winter Simulation Conf.* (WSC), 2006, pp. 1720–1730.
3. B.P. Zeigler, T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, Academic Press, 2000.
4. R. Ewald et al., "Discrete Event Modelling and Simulation in Systems Biology," *J. Simulation*, vol. 1, no. 2, 2007, pp. 81–96.
5. G.A. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, CRC Press, 2009.
6. M. Van Schyndel et al., "On the Definition of a Computational Fluid Dynamic Solver Using Cellular Discrete-Event Simulation," *J. Computational Science*, June 2014, pp. 882–890.
7. T.C. Sudhof and K. Starke, *Pharmacology of Neurotransmitter Release,* Springer, 2008.
8. W.S. Trimble, M. Linial, and R.H. Scheller, "Cellular and Molecular Biology of the Presynaptic Nerve Terminal," *Annual Rev. Neuroscience*, vol. 14, 1991, pp. 93–122.
9. P. De Camilli, "Keeping Synapses up to Speed," *Nature*, vol. 375, 1995, pp. 450–451.
10. F. Benfenati, F. Valtorta, and P. Greengard, "Computer Modelling of Synapsin 1 Binding to Synaptic Vesicles and F-actin: Implications for Regulation of Neurotransmitter Release," *Proc. Nat'l Academy Science*, vol. 88, no. 2, 1990, pp. 575–579.
11. D.T. Gillespie, "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions," *J. Computational Physics*, vol. 22, Dec. 1976, p. 403.
12. G. Wainer et al., "Advanced DEVS Models with Application to Biomedicine," *Proc. Artificial Intelligence, Simulation and Planning Conf.* (AIS), 2007; http://cell-devs.sce.carleton.ca/publications/2007/WJADBDC07/FinalPaper.pdf.
13. R. Goldstein et al., "Vesicle-Synapsin Interactions Modeled with Cell-DEVS," *Proc. Winter Simulation Conf.* (WSC), 2008, pp. 813–821.
14. R. Goldstein and G. Wainer, "Impulse-Based Dynamic Simulation of Deformable Biological Structures," *Trans. Computational Systems Biology*," vol. 6575, 2011, pp. 39–60.
15. M.J. Byrne, M.N. Waxham, and Y. Kubota, "Cellular Dynamic Simulator: An Event Driven Molecular Simulation Environment for Cellular Biology," *Neuroinform*, vol. 8, no. 2, 2010, pp. 63–82.
16. R. Goldstein and G. Wainer, "Simulation of Deformable Biological Structures with a Tethered Particle System Model," *Proc. 32nd Conf. Canadian Medical and Biological Eng. Soc.* (CMBEC), 2009; http://cell-devs.sce.carleton.ca/publications/2009/GW09/Goldstein__TPS__2009-03-17_CMBEC.pdf.
17. D. Shirvanyants et al., "Discrete Molecular Dynamics: An Efficient and Versatile Simulation Method for Fine Protein Characterization," *J. Physical Chemistry B*, vol. 116, no. 29, 2012, pp. 8375–8382.
18. R. Goldstein and G. Wainer, "Simulation of a Presynaptic Nerve Terminal with a Tethered Particle System Model," *Proc. 31st Annual Int'l Conf. IEEE Eng. Medicine and Biology Soc.* (EMBC), 2009, pp. 3877–3880.

Rhys Goldstein is a principal research scientist at Autodesk Research in Toronto, Canada. His interests include discrete-event simulation and its use in the architectural, building science, and biological domains. Goldstein has a MASc in biomedical engineering from Carleton University. Contact him at rhys.goldstein@autodesk.com.

Gabriel A. Wainer is a professor in the Department of Systems and Computer Engineering at Carleton University. His research interests are in modeling, simulation, and real-time embedded systems. Wainer has a PhD in computer science from the Université d'Aix-Marseille III, France. Contact him at gwainer@sce.carleton.ca or via www.sce.carleton.ca/faculty/wainer.